IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

U.S. PATENT APPLICATION FOR

METHODS AND APPARATUS FOR CALL SERVICE PROCESSING

Inventors:

Robert Martin, a citizen of the United States,
residing at 1815 NW 85th Drive
Coral Springs, Florida 33071

Jared Dubin, a citizen of the United States,
residing at 4806 NW 14th Street
Coconut Creek, Florida 33063

Jeff Williamson, a citizen of the United States,
residing at 2614 NW 32nd Street
Boca Raton, Florida 33431

Cathy Tio, a citizen of the United States,
residing at 20661 Bay Brooke Court
Boca Raton, Florida 33498

METHODS AND APPARATUS FOR CALL SERVICE PROCESSING

Reference to Related Applications

5   This application claims priority to provisional applications having Serial No.
60/193,701, filed on March 31, 2000, Serial No. 60/193,807, filed on March 31, 2000,
Serial No. 60/193.809. filed on March 31. 2000. and Serial No. 60/193,700, filed on
and Serial No. 60/ 193,701, filed on March 31, 2000,
March 31, 2000. the teachings of all of which are incorporated herein by reference.
Further, this application is related to United States Patent Application Serial No.

10   09/519,557, filed March 6, 2000, and entitled SYSTEMS AND PROCESSES FOR
CALL AND CALL FEATURE ADMINISTRATION ON A
TELECOMMUNICATIONS NETWORK, the teachings of which are incorporated
herein by reference.

15   Background of the Invention

The invention pertains to telephony and, more particularly, to systems and
processes for call and call feature setup, administration and processing on public
switched telephone networks (PSTN) and other telecommunications networks.

20

Voice telephony (and data communication via telepho ne modem) today is
conducted primarily by "copper loops" or wireless transmission (via base station) from
the user or subscriber to a central office (Class 4/5) switch that initiates the process of
connecting the call to its ultimate destination, whether in the same local exchange

25   handled by the central office, or to a remote central office. The call set-up process
involves a series of "trigger points" wherein the service options available to the caller are
checked against a database that periodically downloaded to the switch from the local
carrier's database (typically relational, i.e., table-based). At the central office switch of
the receiving subscriber, a similar sequence is followed to check such service options as

30   call waiting and call forwarding. Updates of the databases are cumbersome, and adding

1

new services require additional programming and trigger points in an already complex code.

5   The process of call set up outside the local service area further requires an "out of band" communication over the SS7 ("Signaling System 7") network and protocols distinct from the network over which voice traffic flows. It is this system that allows the proper switching of traffic for roaming cellular subscribers and for 800- and 900-numbers. It also provides caller identification. The system, however, is limited in its bandwidth and by the small number of knowledgeable programmers.

10

     The prior art has made advances in the middle or "control" layer of a three-layer network model for voice telephony as part of a trend towards "Advanced Intelligent Networks" ("AIN"). These advances have involved call agents using the SS7 network to control new functionality in call set up and upgrading of the International

15   Telecommunications Union H.323 Internet protocol for coordination of voice, data, and video traffic on the Public Switched Telephone Network (PSTN). For example, Bellcore and Cisco developed the Simple Gateway Control Protocol (SGCP) to improve on H.323 and Level 3 developed the Internet Protocol Device Control (IPDC). The Media Gateway Control Protocol (MGCP) is a merger of SGCP and IPDC sponsored by the

20   Internet Engineering Task Force to migrate SS7 functions into Internet Protocol (IP).

     The current state of the art remains one of limited service options. Any change in service requires the reloading of service maps stored in relational data bases; each new variable for service requires another table, essentially another dimension in the relational

25   database. Each variable is implemented at a stop point in the switching fabric. Switches still require complex programming and cost in the millions of dollars.

     It is an object of the present invention to provide improved systems and methods for call and call feature setup, administration and processing.

30

Another object is to provide such methods and apparatus as reduce the overhead required for implementation or modification of call feature processing.

A related object is to provide such methods and apparatus as facilitate
5   implemenation of call features, whether defined by system designers, service providers or end users.

Still another object of the invention is to provide such methods and apparatus as are facilitate implementation of both complex and routine call feature processing.
10

Yet another object of the invention is to provide such methods and apparatus as can be implemented on a variety of telecommunications platforms.

Yet still another object of the invention is to provide such methods and apparatus
15   as can be implemented at reduced cost and in less time.

Summary of the Invention

The present invention meets the foregoing and other objects by providing, in one aspect, methods in which a compiled representation of a textual description in a mark-up language of operations describing a telecommunication service is generated. Objects, hereinafter referred to as context objects, access this compiled representation, in response to events occurring with respect to a telecommunication service in order to effect the execution of the operations.

Related aspects of the invention provide methods as described above in which the context object is instantiated in response to a boundary event of the underlying telecommunications service or feature. Thus, for example, such methods can result in instantiation of context objects in response to call origination or call termination events.

Further aspects of the invention provide methods as described above in which notification of selected events is passed to an existing context object for processing. The object can respond to such notification by accessing the compiled representation in order to discern one or more operations for execution to further processing of the call feature or service. Notifications which do not result in such further processing can, according to further aspects of the invention, be interpreted as boundary events that result in instantiation of additional context objects.

For example, a system operating in accord with a method of the invention can notify an existing context object of a "hookflash" event. If that event is expected in the normal course of processing of a feature currently being handled by the object, notification results in access of the compiled representation and execution of further operations for the feature. If not, the hookflash is treated as an boundary event and results in creation of a new context object to handle it.

The textual description that embodies the operations can be provided, e.g., by a service provider or subscriber, in a mark-up language such as HTML, XML, or an

4

extension thereof. The operations define a set of rules and actions to be performed in order to provide a telecommunications service. Such a telecommunication service can include, but is not limited to, a call, a call feature, and subscriber or feature administration.

5

According to further aspects of the invention, the compiled representation is implemented in an object-oriented environment, e.g., a C++ object environment. A virtual function, returned as a result of the context object accessing the compiled representation of the operations, can be utilized to invoke one or more core functions required to carry out the telecommunications service.

10

In a related aspect, the textual description embodying the operations is parsed to generate the compiled representation. The context object that accesses this compiled representation can also maintain information regarding present state of an on-going communication service, and be responsive to events occurring with respect thereto.

15

In another aspect, the invention provides the methods described above in which objects, herein referred to as feature object, associated with the context objects embody the compiled representations of the operations and effect the execution of the telecommunication service. The feature objects can receive signals indicative of events occurring with respect to a communication service from the context objects, and effect performance of one or more actions.

20

Utilizing a mark-up language to provide a textual description of the logic specifications indicative of a telecommunication service provides a number of advantages. For example, it places the relatively complex task of defining actions to be performed in response to a myriad of different signals in a graphically depictable format, which can be easily validated. Further, new telecommunications services, e.g., call features, can be designed and validated more conveniently.

25

30

The method of the invention can be implemented, for example, by a telecommunications system that includes a call control module that controls a call processing context associated with a subscriber, and a call feature module that is in communication with the call control module. In response to a call progress event, the call

5   feature module accesses a compiled representation of a textual description in a mark-up language of logic defining a telecommunication service provided to a subscriber to effect the execution of the service.

In a related aspect, the telecommunication system can include a parser that

10   receives the textual description and generates a compiled representation therefrom. The call feature module can instantiate an object that accesses this compiled representation to determine at least one action to be effected in order to provide the telecommunication service.

15   Further aspects of the invention provide systems operating in accord with the methods described above.

Further aspects of the invention provide systems and methods as described above in combination with those summarized and described in co-pending, commonly assigned,

20   "Methods and Systems for Implementing Telecommunications and Other Processes Using an Object Graph Generated from a Markup Language," (U.S. Express Mail Label No. EL835823049US), filed this day herewith, "Telecommunications Portal Capable of Interpreting Messages form an External Device," (U.S. Express Mail Label No. EL835823021US), filed this day herewith, and "Markup Language-Defined State

25   Machine for Use in Call Service Processing," (U.S. Express Mail Label No. EL835823035US), filed this day herewith.

These and other aspects of the invention are evident in the drawings and in the claims that follow.

30

Brief Description Of The Drawings

Figure 1 schematically illustrates a telecommunications system according to the teachings of the invention,

5

Figure 2 schematically depicts exemplary network and external interfaces associated with a telecommunications system according to the invention,

Figure 3 schematically illustrates the interaction of a TSP node of a system of the invention with some exemplary external networks,

10

Figure 4 illustrates that a TSP node of a system of the invention can include a call processing module and a feature processing module in communication with a CSX switch according to the teachings of the invention,

15

Figure 5 illustrates that the call processing subsystem of the TSP node of Figure 4 includes a call control module, a feature control module, and a primitive module,

Figure 6A schematically depicts a call processing model that the call processing subsystem of Figure 5 can implement,

20

Figure 6B schematically depicts a second call processing model that the call processing subsystem of Figure 5 can implement,

Figure 6C schematically depicts a third call processing model that the processing subsystem of Figure 5 can implement,

25

Figure 7A illustrates that a call processed by the call processing subsystem of Figure 5 can be originated from an end-user served by a TSP of a system of the invention and be terminated by an end-user served by the same TSP,

30

Figure 7B illustrates that a call processed by the call processing subsystem of Figure 5 can be originated by an end-user served by a TSP of the system of the invention and be terminated outside the TSP,

5   Figure 7C schematically illustrates that a call request can be extended to a TSP of the system of the invention over a PSTN trunk group or IP route, and be terminated on an endpoint on the TSP,

Figure 7D schematically illustrates that an incoming call request can be extended 10   to a TSP of a system of the invention over a PSTN trunk group or IP route, and can be extended to the next office or TSP over a PSTN trunk or IP route,

Figure 8 is a feature assignment table providing a map of call features to feature logic objects,

15

Figure 9 schematically illustrates that a feature control module of the call processing subsystem of a system of the invention can contain multiple feature logic objects,

20   Figure 10A illustrates a feature logic object corresponding to a call forwarding busy line call feature assigned to an endpoint,

Figure 10B illustrates another feature logic object corresponding to a second call forwarding line busy call feature assigned to the endpoint of Figure 10A as a result of the 25   membership of the endpoint in a group having the second call forwarding feature,

Figure 10C illustrates a call feature object assigned to an endpoint that subscribes to call name delivery and call forwarding variable,

30   Figure 10D illustrates a call feature object assigned to an endpoint that subscribes to call name delivery, call forwarding variable, and denied termination features,

Figure 11A illustrates a few examples of Type I feature logic objects that map triggering events to primitives defined in a primitive module of the call processing subsystem of the invention,

5

Figure 11B illustrates a few examples of Type II feature logic objects that perform one or more transactions, and return the results of the transactions to the event triggering parties,

10   Figure 11C illustrates a few examples of Type III feature logic objects that maintain feature states,

Figure 11D illustrates an example of Type IV feature logic object that can create and/or start another feature logic object,

15

Figure 12 illustrates that dependent feature logics have life spans equal or less than the life span of the associated call context,

Figure 13 is a diagram depicting a logical view of selected aspects of a

20   subscriber's associations with features,

Figure 14 is a diagram providing a detailed representation of selected portions of a core object model representing subscriber associations with features and the resolution of feature interactions,

25

Figure 15 is a diagram illustrating a model that depicts the correlation between instances of a core object model utilized in a system of the invention and the elements of the call and feature definition language of the invention,

30   Figure 16 is a diagram depicting some fundamental elements of a call and feature definition environment of an embodiment of the system of the invention,

Figure 17 is a diagram depicting an exemplary technique by which a compiled C++ object can export its attributes and behaviors to the logic definition environment of the invention,

5

Figure 18 is a diagram illustrating an example of the class hierarchy derived from an Action abstract class defined in a C++ environment in accord with the teachings of the invention,

10   Figure 19 illustrates an exemplary state machine that can determine the parity of a string of 0's and 1's.

Figure 20 is a diagram depicting an approximate schema for logics associated with the state machine of Figure 19,

15   Figure 21 illustrates an object model for a finite state machine,

Figure 22 schematically illustrates exemplary TSP/CSX class 4 and class 5 switches according to the teachings of the invention,

20

Figure 23 illustrates exemplary communications paths within an embodiment of a consolidated TSP/CSX system of the invention,

Figure 24 illustrates a message distribution and transport (software) layer
25   according to the invention interposed between the TSP/CSX application software and underlying network interfaces of a system of the invention,

Figure 25 schematically illustrates a redundant N+1 configuration of a system of the invention having four TSP nodes and three CSX switch elements,

30

Figure 26 schematically depicts another configuration of a system according to the teachings of the invention in which the traffic load for each CSX switch has been partitioned into thirds and distributed equally among three primary TSP nodes,

5   Figure 27 is an overview of an exemplary installation of a TSP/CSX consolidated system of the invention,

Figures 28A – 28B depict a basic application framework for routing and dispatching events in a system according to the teachings of the invention,

10

Figure 29 is an object model illustrating an interface for exchanging messages between a TSP node and another system,

Figure 30A depicts an event trace outlining a process for receiving a message,

15   converting the message into an event, and dispatching the event to an appropriate processing context in a system of the invention,

Figure 30B is a diagram illustrating an event trace that outlines a process of routing an event to an external processor via an external interface,

20

Figure 31 is a diagram illustrating an event hierarchy,

Figure 32A is a table listing exemplary call/feature policy data for a system according to the teachings of the invention,

25

Figure 32B is a table listing exemplary route policy data for a system according to the teachings of the invention,

Figure 32C is a table listing exemplary endpoint/subscriber data set for a system

30   of the invention,

11

Figure 32D is a table listing exemplary call and feature state data for a system of the invention,

Figure 32E is a table listing exemplary equipment/facility data for a system

5   according to the teachings of the invention,

Figure 32F is table listing exemplary equipment/facility state data for a system according to the invention,

10   Figure 32G is a table listing exemplary equipment/facility statistics for a system according to the invention,

Figure 33 is a diagram depicting the correlation between call/feature states in the volatile store and in the persistent store,

15

Figures 34 and 35 depict a state machine for a call waiting feature provided a system according to the teachings of the invention,

FIGURE 36 is a diagram schematically depicting one embodiment of a system of

20   the invention that converts a textual description of a state machine into an object graph,

FIGURE 36A schematically illustrates an object graph created in accord with one embodiment of the invention,

25   FIGURE 36B depicts a methodology for exporting actions from an object repository according to one embodiment of the invention,

FIGURE 37 is a diagram illustrating that an object graph created according to the teachings of the invention can be accessed by an object instantiated to execute a state

30   machine represented by object graph to determine transition from a present state to a new state and any action, if any, to be performed,

FIGURE 38 is a sample XML document containing the description of a state machine for providing a call feature (call forwarding) to a subscriber of a telecommunications service,

5

FIGURES 39 - 46 depict request handlers and feature handlers in accord with one embodiment of the invention,

FIGURE 47 depicts interaction of originating feature classifications in a system

10   according to the invention,

FIGURE 48 depicts interaction of terminating feature classifications in a system according to the invention,

15   FIGURE 49A depicts objects representing a subscriber in an idle state in a system according to the invention,

FIGURE 49B depicts an object model for a system according to the invention,

20   FIGURE 50 depicts processing of an originating call in a system according to the invention,

FIGURE 51 depicts further processing sequences in a system according to the invention,

25

FIGURE 52 depicts processing of a termination request in a system according to the invention,

FIGURE 53 depicts a stable two-party call in a system according to the invention,

30

FIGURE 54 depicts processing of an event, e.g., a flashhook, in a system according to the invention,

FIGURE 55 depicts invocation of a mid-call origination feature in a system according to the invention,

FIGURE 56 depicts three-way calling in a system according to the invention,

FIGURE 57 depicts object configuration upon establishment of a three-way call in a system according to the invention, and

FIGURE 58 schematically illustrates selected processes and their interactions in a system of the invention.

Detailed Description of the Illustrated Embodiment

FIGURE 1 illustrates an exemplary telecommunications system 10 according to the invention having two Telecommunications Service Portal (TSP) nodes 12a and 12b in
5   communication with each other and with a CSX switch 14 to provide a convergent multimedia services environment. In conjunction with the CSX, the TSP 12a/12b offers class 5 and class 4 voice band services. These services can satisfy, for example, the needs of Competitive Local Exchange Carriers (CLEC) and Internet Service Providers (ISP). When coupled with other network elements, intelligent peripherals, and subscriber
10   premise equipment, the TSP offers a solution to the demand for convergent multimedia services environments.

The TSP/CSX system 10 of the invention offers a variety of network and external system interfaces, as depicted in FIGURE 2. For example, the TSP/CSX system includes
15   a Call and Feature Processing subsystem that provides class 4 and class 5 call services and features via the interfaces of the CSX switch with a Public Switched Telephone Network (PSTN) 16. In addition to feature delivery, the system 10 also offers certain feature configuration capabilities through PSTN voice band circuits. Further, the TSP/CSX system includes a Call and Feature Creation and Maintenance Subsystem that
20   keeps the run-time data stores in sync with externally defined calling models and call features. This subsystem also participates in the maintenance of subscriber specific data attributes that may be configured through voice band network interfaces. In addition, a Network Management Subsystem supports common network element operations, administration, maintenance, and provisioning functions covering endpoints, routes,
25   system configuration tables, and provides surveillance of system status and performance metrics. A Billing Subsystem interfaces with external entities in support of billing and usage accounting processes.

Referring to both FIGURES 1 and 2, the TSP/CSX system 10 is situated on a
30   PSTN network 16 and an IP network 18. In combination, the TSP and CSX provide traditional circuit switch based voice band services. The TSP independently provides call

control and feature processing capabilities for IP Telephony Networks (IPTN) assuming multiple roles such as call agent over a MGCP network, proxy server over a SIP network, and so forth.

5   Call Processing Subsystem

Reference to both FIGURES 1 and 3 shows that the exemplary TSP 12a/12b is on the IP network and the CSX is situated on both PSTN 16 and IP network 18. The TSP and CSX jointly provide telephony switching functionality on the PSTN network. On it

10   own, TSP provides call control capability on the IP Telephony Network. For example, TSP can assume multiple roles in various IP telephony networks, such as call agent over MGCP network, and proxy server over SIP network. The TSP can also interface with ATM gateways over an ATM network to provide call control and feature control functions for ATM gateways.

15

On the PSTN side, DS1 trunks terminating at the CSX provide connections between the TSP/CSX and other network elements. These connections include traditional inter-office facilities as well as GR303 based facilities to support subscriber access.

20

Inter-office facilities connect TSP/CSX switches to other end offices and access tandem offices employing a variety of inter-office protocols such as FGD, ISUP, and ISDN/PRI. Each such physical connection between the CSX and another office or network element is called a '*trunk*'. A set of trunks that share common characteristics

25   and traffic load is called a '*trunk group*'. Trunks and trunk groups reside on the PSTN and are managed by CSX. The concept of trunks is encapsulated within the CSX. The TSP has no representation of individual trunks. The TSP does maintain a representation of trunk groups to support call routing, network management, etc.

30   The TSP in conjunction with the CSX, serving as an IDT and RDTs supporting subscriber access facilities, can constitute a class 5 switch. In this configuration, the

16

TSP/CSX system treats the endpoints on the RDTs as its endpoints. The DS0 circuits on these trunks, except the TMC and EOC circuits, are mapped to endpoints and are therefore treated as endpoints. The TSP manages the endpoints.

5   On the IP network, connections between the TSP and other call managers, such as call agents and proxy servers are referred to as '*IP* route's. PC base phones 20 can constitute endpoints. The TSP manages both the IP routes and IP based endpoints network.

10   The TSP treats an IP based endpoint in the same manner as an endpoint on the PSTN. An endpoint can correspond to a subscriber and can be a representation of the subscriber in the TSP. Likewise, the TSP treats an IP route in the same fashion as a trunk group appearing on the PSTN. The TSP manages bandwidth allocation for IP routes whereas the CSX manages the circuits and trunks that constitute a trunk group.

15

           Each trunk group and IP route can serve a number of connections. Each connection to a trunk group or IP route is terminated at a *connection point*. A connection point inherits the attributes of the trunk group or IP route at which the connection terminates.

20

           The TSP also includes the concept of *hunt group*s. A hunt group is a set of trunk groups and/or IP routes that share common characteristics and traffic load. Ultimately members of a hunt group - trunk groups or IP routes - can, but not necessarily, span between IP networks and the PSTN so that calls can be delivered over either network

25   depending upon the network available and other driving factors.

           FIGURE 4 illustrates that the TSP system can include a call processing subsystem 22 and a feature processing subsystem 24. It further illustrates exemplary roles that the TSP can play in various voice/data networks, and it also illustrates its external interfaces

30   with other network entities. The call processing subsystem provides call processing and feature control functions. For example, it can provide call control by supporting a suite

of industry standard protocols, such as SIP, MGCP, and MEGACO, to interface with external switching entities and/or PC phone clients over PSTN/ATM/IP networks. It can further provide the basic call processing functions. The feature control function supports existing PSTN class 4/5 features and enhanced Internet services, such as email service

5   over the voice and packet networks.

The call processing subsystem is responsible for the call processing and switching functionality in the TSP. For example, it can supply basic call processing by providing switched, two-way transmission paths connecting endpoints and inter-switch facilities

10   terminated on the TSP/CSX system. The inter-switch facilities can be, for example, PSTN trunks terminated on CSX and IP routes terminated on TSP. The basic call processing can also include the capability to apply tones, announcements, and alerting mechanisms to end users and inter-switch facilities.

15   The call processing subsystem can also provide resource management. For example, it can manage the call control resources defined in TSP. There are three types of call control resources in the system: endpoints, PSTN trunk groups, and IP routes. Different levels of control are applied to these resources. Endpoints are owned by the subsystem, and the subsystem provides busy/idle handling for them. PSTN trunk groups

20   are owned by switching elements, i.e. CSX. The switching element defines logical circuits representing the physical circuits in the trunk groups and the logical circuits are exposed to the TSP. The call processing subsystem maintains the call states for the logical circuits and tracks the number of in-use circuits in the group, though it is not responsible for the busy/idle handling of physical circuits An IP route consists of a group

25   of logical circuits. The number of logical circuits in an IP route is variable and determined by the bandwidth of the IP route. This subsystem owns the IP routes and provides bandwidth management for them.

Further, the call processing subsystem provides load balancing function for hunt

30   groups. A hunt group consists of a collection of PSTN trunks and IP routes. Multiple load balancing techniques can be applied to maximize the system performance. For

example, a circular hunt mechanism can be implemented. A circular hunt mechanism sequences the trunks/routes in a circular list that has a next route pointer pointing to the next trunk/route to use. When a call is extended out of a hunt group, call processing subsystem uses the circular list to determine the next trunk/route to use, initiates the call

5   on the selected trunk/route, and advances the next route pointer.

Other functions provided by the call processing subsystem include registration and authentication. The subsystem can support dynamic registration and authentication. When an endpoint or trunk is added to the system or removed from it, a registration or

10   de-registration request is initiated by the authority of the endpoint and trunk, which can be the endpoint itself or CSX. An authentication procedure is carried out during the registration and de-registration procedure to assure the system integrity.

Moreover, the call processing subsystem supports multiple multimedia call

15   control protocols, such as MGCP and SIP, and media stream control protocol over heterogeneous networks.

The call processing subsystem can also provide support for real-time and post-processing billing model. Real-time Call Detail Record (CDR) is generated and is stored

20   in a persistent data area during runtime.

The call processing subsystem also supports class 4 and class 5 basic and supplementary telephony services on POTS line over GR303 interface. Appendix A provides a list of class 4 and class 5 features that are supported. In addition, the

25   subsystem supports TCAP applications. For example, it supports 800 number service, Local Number Portability (LNP), and Calling Name Delivery (CND) applications over TCAP/SS7.

The call processing subsystem of the invention can employ an Advanced

30   Intelligent Network (AIN) architecture. It can further incorporate new three-tier call processing architecture technology to offer switching calling features and new IP services

on one single platform. The call model of the invention provides a number of advantages. For example, improper feature definitions do not prevent the TSP/CSX system of the invention from meeting call processing behavior that is expected from peer level network elements. Further, call processing level response times are subjected to

5　more stringent requirements, governed by specification, than feature level response times which are governed by human perceptions of "quality of service." The TSP supports portability by operating at the "network" level, e.g., via a SIP level interface, as well as at the "call processing" level, e.g., at the level of Parlay. In addition, feature processing and feature interaction are simplified by employing an AIN basic call model, where each call

10　constitutes two half calls, origination and termination. The origination half call deals with the origination features and termination half call deals with the termination features.

The system of the invention allows separating feature processing from basic call processing. For example, the TSP can provide call control, feature control, and/or a

15　combination of call control and feature control. An interface between the call processing entity and feature-processing entity can be asynchronous or synchronous. A number of standards are defined, for example, by VoIP industry, for such an interface. PARLAY and JAIN are two such commonly used standards that define extensive sets of APIs. As the VoIP technology evolves, PARLAY and JAIN continue to change. The system of the

20　invention fully supports industry standards on this interface in order to inter-operate with equipment from various vendors. Other API, such as private proprietary API, can also be defined for providing communication between the feature processing and call processing subsystems of the system of the invention.

25　FIGURE 4 illustrates schematically the various roles that the TSP can play in various voice/data networks and its external interfaces with other network entities.

FIGURE 5 illustrates that the call processing subsystem of the TSP can include a call control module 26 for managing basic call processing, a feature control module 28

30　for handling feature processing, and a primitive module 30 that contains a set of call control methods, such as playTone, utilized by feature and call control modules.

20

FIGURE 5 also illustrates the data objects (databases) utilized by these modules and their collaborations.

5   With the separation of call control and feature control functions, the databases for supporting each module are also separated. The exception is the CPML policies and route table. The CPML policies are for the feature processing, and are accessed by the feature control module only. The routing function can be viewed as the basic call processing as well as feature control. Therefore, this table can be accessed by both call and feature control modules.

10

The call processing subsystem of the invention can operate in three models to provide different levels of call and feature control and/or management. FIGURE 6 schematically depicts Model 1 that offers a complete call and feature control solution. In this model, the call control module receives external call control events. If the event

15   triggers feature control, call control module generates a triggering event and sends it to the feature control module. The feature control module processes the event, and issues a command, by invoking, for example, a primitive in the primitive module. Alternatively, the command is sent down to the call control module, which in turn calls a primitive in the primitive module. In this model, all external events from the switching elements are

20   channeled through the call control module. However, both feature control module and call control module can communicate with external switching elements through the primitive module.

FIGURE 6B schematically depicts Model 2 that offers call control capability.

25   Further, FIGURE 6C schematically depicts Model 3 that offers feature control capability and can be utilized to provide calling features/services to foreign switching elements, such as featureless call agents and gatekeepers.

The primitive module 30 provides a call control library that contains a set of APIs

30   that the feature control and call control modules can invoke to interface with the external world. These APIs can define generic call control functions, such as makeCall,

acceptCall, answerCall, releaseCall, mergeCall, etc. A frame work API set provides the supporting capabilities necessary to ensure system security and manageability. Some examples include registration, authentication, and discovery procedures.

5   A call control API set is utilized to control the switching elements. Although this API set is primarily invoked by the control module, it includes a subset that can be called by the feature control module as well. The methods in this API subset can be called either directly by the feature control module or indirectly through the call control module. This API set can run on multiple call control protocols simultaneously. The selection of 10   the protocol is done at the initialization phase. Further, a subset of call control API can be exported to CPML+ to be accessed by end users in order to control the call treatments.

      The primitive module further includes a set of feature control APIs, called by the feature control module, for supplying the utilities for controlling the behavior of a call, 15   with each call having up to 2 legs. This API set provides enough functionality to allow call routing and call leg management for existing circuit and packet telephony networks. Some examples of this API set include, but are not limited to, `enableCallNotification, disableCallNotification, getCallState, callEventNotification, createCallLeg, deleteCallLeg, attachCallLeg,` 20   `detachCallLeg, routeCallLegToAddress,` and `releaseCall.` A subset of the feature control APIs can be exported to CPML+ to be accessed by the end users for the creation and modification of calling features.

      The primitive module also includes a set of service API's for accessing external 25   services, e.g., 800 number service, calling name server, email server, and so forth. This API set encapsulates protocol specifics to relieve the invocation entities from dealing with that level of detail. SMTP and MIME , called by feature control module, are examples of members of this API set. A subset of the service APIs can be exported to CPML+ and accessed by the end users for the creation and modification of calling 30   features.

22

The call control module provides the basic call processing functionality for TSP. Such call processing functionality can include, for example, called ID analysis and routing, intercept treatment (tone and announcement), overload handling, detection of feature triggering events and reaction to them, generation of CDR, and generation of call

5   related traffic counts.

The call control module employs a call model, herein referred to as TSP call model, that is derived from the AIN Basic Call Model (BCM). By AIN definition, a typical call has an Originating Basic Call Model (OBCM) at the switch of the caller side,

10   and a Terminating Basic Call Model (TBCM) at the switch of callee side. Each basic call model defines a set of points in call (PICs) that correspond to important states in a call. Associated with each PIC is a set of Detection Points (DPs). A DP is associated with a set of triggers each of which specifies the conditions under which an AIN feature can be invoked.

15

In the TSP call model, each call consists of one OBCM and one TBCM, where the originating features/services apply to the OBCM and terminating features/services to the TBCM. An interface between OBCM and TBCM can be asynchronous, and can utilize an industry standard call control protocol. A BCM is a state machine driven by

20   the call progress events. Any call progress event with which a feature can be associated is a triggering event. When a triggering event occurs, a notification is sent to the feature control module. The feature control module will take appropriate actions in response to the notification. The details of interface between the call control and feature control modules will be discussed in more detail below. The BCM exists in the form of an

25   endpoint or connection point, depending upon the nature of origination/termination point of the call.

The call processing subsystem processes the calls arriving at the TSP. As a class 5 switch, a call can be 1) originated from an endpoint, and terminated to an endpoint –

30   intra-office call, 2) originated from an endpoint, and terminated to a trunk group or IP route – inter-office call, or 3) originated from a trunk group, and terminated to an

endpoint – inter-office call. As a class 4 switch, a call is originated from a trunk group or IP route and terminated to a trunk group or IP route. These are the four types of call scenarios supported by this call model.

5   In case 1, schematically depicted in FIGURE 7A, a call is originated from an end user served by the TSP and is terminated to an end user served by the same TSP. In case 2, depicted in FIGURE 7B, an end user, served by the TSP, originates a call and the call is terminated outside of the TSP. For example, the call is extended to another PSTN office or TSP through a PSTN trunk group or IP route. A connection point is allocated 10   based on the trunk group or IP route to serve the egress leg of the call.

          In case 3, depicted in FIGURE 7C, an incoming call request is extended to the TSP over a PSTN trunk group or IP route, and the call is terminated on an end point on the TSP. In this scenario, based on the trunk or IP route a connection point is allocated to 15   serve the ingress leg, and an end point is allocated to serve the egress leg.

          In case 4, depicted in FIGURE 7D, an incoming call request is extended to the TSP over a PSTN trunk group or IP route and the call is extended to the next office or TSP over a PSTN trunk group or IP route. 20

          The call control module maintains a set of run-time data objects to support the basic call processing operation. For example, the call control module can create an endpoint status object when a call is initiated. The endpoint status object can contain all the endpoints statically configured and dynamically registered to be managed by the TSP. 25   Each endpoint that is dynamically registered with the TSP is created with the status set to in service upon the successful registration, and is removed when the de-registration is completed. For the endpoints statically configured to be managed by the TSP, upon creation of this data object, the call control module accesses the system configuration data object to retrieve the status of the endpoints belonging to the TSP, and creates 30   endpoints in the endpoint status object.

Another data object is a connection point status object that is also created when the call control module is initialized. This object keeps track of the in-use connection points in a PSTN trunk, or IP route.

5   Further, a call object is created when a call is initiated and deleted when the call is released. Each call has 2 call objects, one for the origination (ingress leg), and the other for the termination (egress leg). An endpoint or connection point can own a call object.

The TSP infrastructure provides a set of mechanisms for the storage of the
10   runtime data objects, discussed below, that the call control module can utilize in order to support the system performance and reliability requirements.

Referring again to FIGURE 5, the call processing subsystem also includes a feature control module that provides supplementary calling features and services to the
15   endpoints and PSTN-trunks/IP-routes served by call managers, for example, the call control module. The call feature module provides PSTN class 5 and call 4 features. In addition, new features can be created by the carriers and ISPs. In particular, TSP exports a set of functions, such as access 800 number data base on SS7 network, sending emails, callback utility, to CPML+. The carriers/ISPs can utilize these functions to create new
20   services. These features/services are available on system level, and can be assigned to the entire system, individual groups, or end users. Charges may be expected either in usage or subscription basis. If the introduction of the new services and features changes the behaviors of the existing features, the creators of the features are responsible to specify the feature interactions.
25

The end users can also create new features for their own use, for example, by utilizing the capabilities of CMPL+. Typically this type of features are offered at no cost to the end users, unless the new features access the information from the system or utilize system resources, such as TCAP services.
30

The first three types of features described above are system-wide features, which are available to all endpoints/connection-points on the system. The last types of features are end user features that are only available to the creators of the features. The system-wide features can be assigned to the whole system, to individual groups, and to

5   endpoints/connection-points. A group is defined as a collection of endpoints/connection-points, served by the same TSP, that share the same set of features. A group can be a business group, a residential group, a trunk group, etc. The members of a group have access to the group features. The end user features, however, can only be assigned to the endpoints/connection-points.

10

In the feature domain, there are system-wide features, such as a usage sensitive three-way call feature, that are available to all endpoints/connection points on the system. There are also group-level features, such as a call pickup feature, that are available to the members in a group, and endpoint/connection point level features, such as a call waiting

15   feature, that can be subscribed by all endpoint/connection points on the system.

An endpoint/connection-point can have all levels and types of features assigned to it, thus necessitating management of interactions among various features. The feature control module can handle a number of different feature interactions. Two features can

20   co-exist without impacting each other's behavior (no interaction). Alternatively, two features may be mutually exclusive. Two mutually exclusive features are not allowed to co-exist. For instance, call waiting and multiline hunt service cannot be assigned to the same endpoint simultaneously. A feature rule table is defined to specify if any two features in the system are mutually exclusive. When a feature is assigned, this table is

25   utilized for validating the feature.

In some cases, a feature may be assigned to an endpoint/connection-point multiple times through different channels. In such cases, one feature would override the others. In particular, the system level, group level, and endpoint/connection-point level features

30   have different configurable priorities. At the time of feature assignment, the feature

interaction is resolved and a decision is made regarding which feature will take effect based on these priorities.

5   In cases in which two different features co-exist, one feature can override the other. For instance, when denied origination and call forwarding variable are assigned to the same endpoint, denied origination overrides the call forwarding variable. This interaction is also handled based on a comparison of the priorities of the features. As discussed in more detail below, each feature is associated with a feature logic object that defines the feature behavior. More particularly, a feature has one base feature logic object 10   that defines the feature logic when the feature operates alone, and can have multiple modified feature logic objects that define the feature logic when the feature operates with other features.

A feature assignment table 30a, shown in FIGURE 8, maps features to feature 15   logic objects. The feature assignment table is referenced to populate feature objects for the endpoints/connection-points when features are created and assigned. The assignment table contains the feature logic objects defined in the system. Further, the assignment table is indexed by a feature mask, which is a bitmap of the features defined in the system. When a feature is present, the bit corresponding to the feature is set to 1.

20

Thus, the feature control module handles feature interactions by implementing at least four mechanisms, namely, assigning feature priority to each individual feature, providing a system wide feature rule table, providing a system wide assignment table, and feature logic encapsulation. This allows resolving feature interactions/conflicts when 25   features are created and assigned, thus improving the runtime performance

A more preferred embodiment of the invention provides feature classifications and resolves feature interactions in accord with the teachings provided below in connection with the description of an additional embodiment, for example, in a section 30   entitled "Service Provisioning." In such an embodiment, a request handler processes

features in accord with their pre-defined classifications to ensure that no conflict betweeen various features occur.

In the feature domain, a feature is defined as a (*event, behavior, data*) triplet;
5   where *event* indicates a condition when the feature is invoked, *behavior* is a sequence of actions to take place when the feature is invoked, and *data* is the information used by the feature *behavior*. The triplet constitutes a Feature Logic Object (FLO). The features assigned to an endpoint/trunk/IP-route are stored in a feature object. Every endpoint/trunk/IP-route has one feature object. A feature object contains multiple FLOs
10   and one active calling feature stack, as shown in FIGURE 9.

An FLO describes the (*event, behavior, data*) triplet of the feature, and defines the administrative data associated with the feature. The association between a FLO and a feature object is the event. Although a feature object can have multiple FLOs, the
15   associations are unique within a feature object. An FLO can include an identifier, which is the feature name uniquely identifying the feature within a TSP system. For system wide features, an identifier is a mandatory field that can be updated only by authorized personnel. An *event* specifies the condition for occurrence of a feature. There are two types of events; base events and extended events. The base events are the typical call
20   progress events in the basic call model and the acknowledgement events. Call progress events can include, for example, origination side: offhook, dialComplete, remoteAlerting, and remoteAnswered; termination side: seizure, alerting, and answered; both: hookflash, localRelease and remoteRelease.

25   Acknowledgement events are returned to the feature object upon completion of a task, initiated by the feature object. For instance, the feature object can instruct a call object to start a 30-second timer. Upon the expiration of the timer, the call object returns an acknowledgement event to the feature object.

30   The extended events are a set of conditions associated with the base events. For instance, the 800NS extended event is defined as the detection of the dialComplete event

and the dialed number that starts with 800. With the extended events, some of the feature control logic is shifted to the call control objects to reduce the traffic between feature domain and call control domain. The flexibility of this technique provides better system performance, less exchange of messages between the two modules, and the ability to

5   introduce new events.

An *Event Definition* is present when the event is an extended event. The event definition specifies the construction of an extended event. For example, the event definition of 800NS is "dialComplete event + prefix of dialed number equals to 800".

10

As mentioned above, the features are prioritized in order to resolve two types of feature conflicts: same feature assigned multiple times to an endpoint/connection-point, and two different features assigned to an endpoint/connection-point where one overrides the other. A priority attribute is divided into two fields: group level field and feature

15   priority field. The group level field resolves the first feature conflict, namely, the multiple assignment of the same feature. It specifies the feature priority based on the ownership of the feature. A feature can be a system level feature, group level feature, or endpoint/connection-point level feature. Each of them can have different priorities and different groups can have different feature priorities. The group level field is configurable

20   and can be controlled by the system operator.

The feature priority field settles the second feature conflict, e.g., one feature overriding another when two features co-exist. New feature interactions are controlled and specified by the feature creators.

25

A feature object can include an ownership attribute that identifies the owner of the feature. This field is used to propagate the updates on the features, and to remove the FLOs from feature objects when features are deleted.

29

A method of a feature object refers to a sequence of actions to take place when a feature occurs. It can be a simple command, such as route the call, a complete state machine, or another FLO.

5   Feature specific data is saved only when a feature is subscribed or activated. For example, the last incoming caller number is only saved when an end user subscribes to the Automatic Recall feature.

The stack is utilized when more than one feature can be invoked at the occurrence
10   of an event, and/or same feature is assigned to an endpoint/connection-point multiple times. The FLOs of the features that should be invoked when an event occurs are put in the stack. When an event is detected, the associated FLO is retrieved. If the activated feature stack of the retrieved FLO is not empty, the FLOs in the stack will all be executed. The following examples better illustrate this mechanism.
15

In Example #1, schematically depicted in FIGURE 10A, an endpoint A subscribes to Call Forwarding Busy Line feature (CFBL). The feature is activated with the forwarded number set to 1234. The feature object for endpoint A will have an association to the CFBL1-FLO.
20

With reference to FIGURE 10B, the endpoint A may be a member of a group XYZ to which the system administrator may also assign the CFBL feature in order to forward all incoming calls to 6789 when the rung station is busy. Thus, a new CFBL FLO, CFBL2-FLO, will be associated with the endpoint A. If a group feature has a
25   higher priority than an endpoint feature, CFBL2_FLO overrides CFBL1-FLO. Hence, CFBL2-FLO will be put in the activated feature stack and will be executed when an incoming call terminates on endpoint A, and endpoint A is busy.

In Example #2, depicted in FIGURE 10C, an endpoint B subscribes to calling
30   name delivery and call forwarding variable (CFV), and call forwarding variable is activated. If the endpoint B also adds the Denied Termination feature to its feature

30

profile, it will have three features that co-exist at the seizure event, as shown in FIGURE 10D. According to LSSGR /01-02-0500/; when CFV and denied termination co-exist, the latter should take precedence.

5   The system of the invention provides at least five types of FLOs. The categorization is based on the functions that the FLOs provide. An FLO belonging to type I maps a triggering event to a primitive defined in the primitive object. That is, the method of the FLO is a primitive from the primitive object. The feature data, when present, is the input to the method. Upon completion of the primitive execution, the FLO

10   is completed. FIGURE 11A depicts a few examples of Type 1 FLOs.

         Another set of FLOs, herein referred to as type II, are transaction oriented. These features perform one or more transactions, and return the results of the transactions to the event reporting parties, e.g., the call control module. The transactions can be, for

15   example, TCAP database queries, sending an email, sending a fax, etc. The transaction results are the results of the database queries, or the acknowledgements of the completion of the transactions. This type of FLO requires a transaction state. FIGURE 11B illustrates some example of type II features.

20   Another type of FLOs, herein referred to as type III, maintain feature states. State machines are utilized to implement the complicated behaviors of these features. That is, sets of dynamic events are typically defined as part of a calling feature that is enabled when the feature is activated. These dynamic events are the driving events of the feature state machine. They are disabled once the feature is de-activated or terminated. For

25   example, when three-way feature is activated, the dynamic events hookflash, remoteRelease, and localRelease are enabled. Once the three-way call feature is terminated, these events are disabled. FIGURE 11C depicts some examples of type III features.

30   An FLO, herein referred to as type IV, can create and/or start another FLO. The activations and deactivations of features are of this type. There are two sub-types in this

category. One sub-type relates to creating an FLO and associating the FLO with the feature object. For instance, Call Forwarding Variable (CFV) is defined as (seizure, routeCall, forwarded number). When an end user activates the CFV feature, the result of the feature is the creation of the CFV-FLO and its assignment to the end user's feature

5   object with the seizure event. The other sub-type relates to starting the FLO. For instance, when an end user activates the Automatic Recall (AR), the AR-FLO state machine is started. FIGURE 11D illustrates an example of a type IV FLO.

Another type of FLOs, herein referred to type V, are a combination of above types

10   I-IV FLOs. The FLOs of type I through type IV encapsulate feature behaviors of a single feature, or multiple features behaving as one feature when running together. There are situations that multiple features are defined at the occurrence of one event A determination of which features to invoke can depend on the data accompanied with the event and/or the feature priority and interaction. The user data associated with events can

15   be utilized to determine the feature invocations. This can be implemented, for example, by (1) associating a condition with an event (i.e., defining an extended event) or (2) with an executable script. The script uses the user data to navigate the feature invocation process. The method in a type V FLO can be a reference to an executable script. For example, if there are no extended events defined in the TSP, the FLO of dialComplete

20   event will have to use the dialed digit to determine if 800 services, 900 blocks, or other features should take place.

When multiple features exist at one event, they can all be invoked as long as they do not interact with one another. These features should have the same feature priority.

25   For example, the calling name delivery and distinctive ringing features, defined at the seizure event, can be invoked when seizure event occurs. If the features do interact with one another, and one overrides the other, they will be assigned different priorities.

If a feature is invoked and it does not terminate at the end of the invocation, the

30   FLO of the feature is put in the stack waiting for the occurrence of further events to complete the feature. There can be multiple FLOs on the stack for an

endpoint/connection-point. When the feature is completed, the FLO is removed from the stack.

5   As a consequence of the invocation of a feature, the call control object may need to monitor new sets of events. These new events are herein referred to as dynamic events. The dynamic events are to be enabled only after a feature is invoked, and are disabled once the feature is terminated. A dynamic event is not associated with the endpoint's feature object. It is rather an event in the feature state machine.

10   **Call and Feature Definition Language**

      The present invention employs a mark-up language, such as CPML (a derivative of XML), to define a variety of logic elements, e.g., finite state machines, such as, call processing models, calling features, re-usable calling feature elements, etc. A preferred
15   embodiment of the invention employs CPML call and feature definition language and its supporting environment.

      A call and feature definition language utilized in the system of the invention should provide an effective and efficient means of developing and deploying network
20   based multimedia services, for example, class 5 and class 4 call services in conjunction with a circuit switch. Multimedia features within the scope of the language must include the entire spectrum of standard subscriber services[1], e.g., call waiting, call forward when busy, etc., as well as so-called next generation or convergent services, e.g., packet based voice service with common subscriber voice services. The language should also allow
25   concise and efficient expression of simple services, such as call forwarding, while providing sufficient expressive complexity to define complex services, such as call waiting and three-way calling.

---

[1] This is the collection of calling services commonly referred to as "class 5" and "class 4" services.

Secondarily, the language must facilitate the leveraging of web-based technologies in support of a variety of peripheral applications such as subscriber provisioning and maintenance, "data browsers", feature editors/generators, feature validation and testing tools, etc. The language should also support feature definition and deployment within a

5   matter of days, and should simplify overall application development by reducing dependence upon compiled code.

The call and feature definition language should make it possible to define sophisticated media services, e.g., voice calling features, without traditional software

10   requirements: analysis, development, testing, and deployment cycle. In this regard, the language must provide the ability to define features without a "software compile." The run-time environment supporting the language defined calling features must also be sufficiently efficient to meet performance, reliability, and quality of service requirements for a reasonably large subscriber base.

15

The call and feature definition language must be able to define a variety of logic elements including call-processing models, calling features, re-usable calling feature elements, etc. A run-time core, consisting of a compiled object model supported by an object-oriented database, provides the fundamental capabilities (primitives) from which

20   media services, e.g., call feature, are defined. These primitive capabilities are exported, i.e., made accessible, to externally defined logics that comprise call models and call features. Each logic element is therefore defined in terms of well-known behaviors and attributes exported by the run-time core.

25   In the system of the invention, the call and feature definition language is used, in effect, to define finite state machines. The mark-up languages CPML/CPML+, derivatives of XML, are well suited to this approach. In one aspect, the language provides a standardized interface between feature definition applications (GUI desktop applications) and the service delivery systems. The XML-based languages, such as

30   CPML, have the additional advantage that they are human readable as well as machine readable.

34

As discussed previously, the simplest call features are stateless. That is, the behavior of such features may be described as a simple reaction to an event because the event is closely bound to a specific state, i.e., that there is a unique set of circumstances

5   that yields the particular event. Tight coupling between an event and a specific state eliminates the need to represent the state explicitly in the feature logic of such call features. However, certain features are sufficiently complex to require explicit representation of the state. This is particularly true when an event may be coupled with several states, i.e., when the circumstances that yield a particular event are non-unique.

10   Under these circumstances, the state must be represented explicitly.

The logic elements defined by the call and feature definition language of the invention can include embedded parameter values. Such parameter values are accessible directly because they are embedded in line with the logic elements. This alleviates the

15   need to navigate through the run-time object model in order to access the appropriate values. Further, embedding values within logic elements provides a more efficient expression of the feature logic. Embedding the values of attributes associated with a carrier within a logic element adds no additional storage overhead to the system because attributes associated with the carrier are in effect global values. However, embedding

20   values within a logic element for attributes associated with a subscriber introduces additional storage overhead since there must be a distinct instance of the logic element for each subscriber that utilizes the feature. This affects both persistent store and processor memory resource consumption. Embedding attribute values within logic elements may also lead to data duplication, if two or more logic elements use the same

25   attribute, thus complicating maintenance of the run-time data store.

In an alternative model, the embedded values of attributes constitute default values, assigned by the feature definer, that apply when there is no other specific value available within the processing context. In this model, an individual subscriber has the

30   opportunity to define values for these attributes, but is not required to do so. When the

35

subscriber does not provide an explicit value for one or more of these attributes, the embedded, i.e., default, values apply. This offers additional flexibility to the subscriber.

5   The call and feature definition language allows defining associations between logic elements. This capability supports higher order capabilities such as defining re-usable logic elements and parameterized meta-features. For instance, conducting a dialog between a feature and a subscriber using voice prompts and DTMF responses is a feature generic capability. That is, the logic for such a dialog can be defined without regard to the specific prompts to be used or the subscriber's responses. The fundamental concept

10   of a "prompt and response" dialog is common to and therefore re-usable by a variety of calling features.

1. Specify invocation of associated logic elements including the passing of parameters from calling logic to called logic.

15   2. The ability to specify the circumstances wherein one logic element activates, or *calls*, an associated logic element is fundamental to re-using feature generic logic elements.

The language also allows defining extensions to the core object model. In particular, the language, along with appropriate run-time support, includes some

20   capability for extending the core object model by introducing new data objects. These extensions are referred to as data object because they hold values required by the associated feature logic. This advantageously alleviates the possibility that the core object model may not completely anticipate the specific needs of all features to be offered.

25

Call models and calling features are essentially event driven. Thus, the language provides the ability to associate behavioral responses with events. This includes specifying the source or sources of a particular event and whether or not the event is synchronous or asynchronous.

30

36

The language can define features based upon the attributes and behaviors that comprise the core object model as well as extensions to the model, based upon available attribute types, defined by the language itself. The feature definition environment preferably includes a representation of the core object model in order to provide the most

5   complete expressive capability. Further, the language allows identifying associations between run-time objects. Utilizing associations between core objects allows exploiting the core model efficiently.

The language also Identifies means of navigation of associations between run-

10   time objects. In order to access behavior and attributes through associations, the language includes means for specifying navigation from object to object using known associations.

A logic element must be able to express interest in a particular event in order to

15   respond to the event. The underlying core objects can propagate events along these lines of interest in order to ensure correct behavior.

The language can also identify whether a logic element is dependent or independent of a particular run-time processing context. Each feature or call model logic

20   executes with respect to a processing context. Processing contexts, which are part of the core object model, are the basis for routing and dispatching events and messages as well as gaining access to processing threads. There are two fundamental processing contexts within the core object model—the call and feature classes. The call class provides the processing context for call models. Call model logic elements are always subordinate to

25   a specific call context since the life span of a call and a call context are always coincident. However, feature logics may be dependent upon or independent of a call context.

As depicted in FIGURE 12, dependent feature logics have life spans no greater

30   than the associated call context. Therefore, a dependent feature logic element may be integrated into the associated call context or it may execute independently using a

37

dedicated feature context. Independent feature logics may have life spans that exceed the associated call. Indeed, certain features may span several calls. Independent feature logics must therefore use dedicated contexts in order to facilitate the appropriate life span regardless of the life spans of the associated call or calls.

5

The use of a markup language as a call and feature definition language in the system of the invention provides a number advantages. For example, markup document parsers are typically dependent only upon the structure of a document as specified by the document type definition. In particular, parsers are independent of the contents of the

10   documents. Hence, introducing the appropriate document type definition and related applications effectively introduces a new document type into the system. That is, it is not necessary to define, implement, and test a new parser. The parser can therefore be re-used by a variety of different applications.

15   A validating parser may assist with context validation by insuring that a markup-based document adheres to the specified structure. It may also assist with certain types of data element validation, e.g., it may support validation of enumerated or other constrained data types.

20   Further, markup languages tend to support schema flexibility. A markup-based language can specify a variety of data structures including hierarchical structures that are difficult to implement via traditional line oriented ASCII files. Using the capabilities inherent in the call and feature definition language, it is possible to define a variety of log file formats, such as CDRs, system logs, and so forth, so long as the requisite attributes

25   exist in the core object model or in an extension to the core model, or can be derived from existing attributes, and further, accessor methods are available. Furthermore, given that one logic element can reference and invoke another, using the language to define and produce log files provides substantial flexibility.

30   In addition, markup languages provide a standard and flexible way of capturing and representing data entities that are shared among heterogeneous execution

environments. Different implementation languages may be used for different applications. While the core run-time system calls for a compiled object model, peripheral and front end applications may be implemented by utilizing a more convenient language such as Java.

5

The control and feature definition language and its supporting environment can facilitate and accelerate development of peripheral applications, such as a subscriber provisioning and maintenance application by supporting languages and development tools optimized for this purpose. Furthermore, maintaining data repositories, such as the

10　subscriber database, in markup form may simplify development and maintenance of peripheral "back office" applications, as well as interfaces to external systems.

FIGURE 13 is a diagram depicting a logical view of selected aspects of subscriber associations with features. In this diagram, a cloud labeled *CPML FLN* represents a call

15　feature defined by the language, and a cloud labeled *CPML FDN* represents a feature data object defined by the language. Ovals represent object instances within the run-time system. Objects labeled *FLON* represent feature logic objects; objects labeled *FDON* represent feature data objects. Further, objects labeled *SGN* represent subscriber group objects; and objects labeled *SN* represent individual subscribers. Lines and arcs represent

20　associations or relationships between the entities connected to the ends.

Feature logic objects correspond to feature logic definitions. They are in effect "compiled" representations of the logic constructed from object instances that correspond to the logic elements used by the definition language. These objects are used by the run-

25　time system to direct and control the core object model in order to deliver the desired feature behavior.

Feature data objects, as shown in FIGURE 13, represent instances of the "schema" defined by the feature definition elements. These objects, labeled *DEXTN*,

30　extend the core object model "schema" (the *S* and *SG* ovals represent the core object model). Although there is an association between the language-based feature logic and

39

feature data specifications, which indicates the feature logic's dependence upon the specified data, this association is not implemented directly within the run-time system. Instead, this association is implemented indirectly through the subscriber objects. This allows feature logic objects, as shown here, to remain context free. Feature logics can be

5   instantiated as singletons, thus reducing storage overhead so long as they remain context free.

This model also depicts the three mechanisms for associating features with subscribers. Each subscriber is associated with the carrier features provided to all

10   subscribers, e.g., *FLO1*. Two of the subscribers, *S2* and *S3* are members of group *SG1* and therefore are associated, indirectly, with features *FLO2* and *FLO3*. Because these subscribers are associated with a feature (*FLO3*) for which there is a feature data extension to the core object model, these subscribers have been provisioned with instances of the appropriate feature data object, namely, *DEXT3*. Subscriber *S3* has an

15   individual level subscription to feature *FLO4*. Accordingly, this subscriber has also been provisioned with an instance of *DEXT4*.

FIGURE 14 provides a detailed representation of selected portions of the core object model. This model represents subscriber associations with features and the

20   resolution of feature interactions. A subscriber, which uses 0 to many endpoints (media devices), may subscribe individually to 0 or more features. A group, consisting of 0 to many subscribers, may subscribe to 0 or more features. A carrier, which provides service to 0 to many subscribers and provides service to 0 to many subscriber groups, may offer 0 or more features.

25

A feature may exclude another feature, may override another feature, or may interact with another feature. When an exclusion relationship exists between two features, feature provisioning must resolve this conflict so that one and only one of the two features is available to the subscriber's endpoint (at run-time). When one feature

30   overrides the other, the run-time system must ensure that the two features are not executed coincidentally and that the higher priority feature is preferred. When two

features interact, a third feature, which resolves the interaction, must be defined to substitute for the conflicting individual features. The available features association between a subscriber's endpoint(s) and features represents the resolution of these interactions.

5

FIGURE 15 illustrates a model 32 depicting the correlation between core object model instances and the elements of the call and feature definition language. One or more endpoint objects are associated with the subscriber object. A finite state logic instance, representing the call model, is associated with each endpoint and is consistent with the endpoint type. This object controls each call-processing context that becomes associated with the endpoint. Further, a set of references to available features is also associated with each endpoint. Each feature is associated with a finite state logic object and may be associated with a feature data object. The finite state logic object controls each feature instance (the feature data instance). A component of the subscriber object, when present, contains the necessary subscriber specific attribute values.

A call processing context is always associated with an endpoint. A feature-processing context may be associated with a call processing context and/or may be associated directly with an endpoint. A feature-processing context representing a dependent feature may be associated with a call-processing context without being associated with an endpoint. An independent feature must be associated with an endpoint.

FIGURE 15 also depicts the correlation between objects within the run-time environment and external call and feature definition components. A subscriber page describes a subscriber instance (object). In turn a subscriber page is an instance of the subscriber class, i.e., an instance of the subscriber document type. The subscriber class (document type definition) defines the name and type of each attribute as well as the name, type, and parameters of each behavior exported by the subscriber object class. The document type definition is used to capture and validate subscriber attributes and to validate logic instances that reference, i.e., depend upon, subscriber behavior and

attributes. Similar relationships exist between other objects and external documents (pages).

Call and Feature Definition Environment

5

FIGURE 16 depicts some fundamental elements of a call and feature definition environment of an embodiment of the system of the invention. Three fundamental document types, based on XML, for example, are present, namely, an object meta-class, a provisionable object meta-class, a logic class.

10

The object meta-class is the document type definition for describing non-provisionable core object classes. A non-provision able class is one whose instances may be instantiated only during run-time call and/or feature processing. Examples of non-provisionable classes include call-processing context, feature processing context, event

15   classes, and so forth. These class definitions specify attributes and behaviors exported by the core object model that may be used to define call processing and feature logics.

The provisionable object meta-class is the document type definition for descriptions of provisionable core object classes. A provisionable object class is one

20   whose instances are provisioned, i.e., a class whose instances are defined and maintained by a peripheral application, and then imported into the core object model. Examples of provisionable object classes include, for example, subscriber and endpoint. These class definitions support the provisioning and maintenance applications as well as define the attributes and behaviors exported by the core object model for call and feature processing

25   logic definition. In one embodiment of the invention, provisioned instances are maintained in markup form to support various peripheral applications such as subscriber maintenance, bill processing, and so forth. Maintaining a repository of markup-based data eliminates the need to extract data from the run-time data store to support these applications.

30

42

The logic class is the document type definition for descriptions of call and feature processing logic elements. As shown, logic descriptions are described with respect to object meta-classes and provisionable object meta-classes. In other words, class descriptions derived from the object meta-class and provisionable object meta-class are

5   referenced by logic elements derived from the logic class. These relationships support definition, through specification of attributes and behaviors, and validation, by prohibiting the use of non-existent attributes and/or behaviors in the definition of a logic element. This capability also provides a certain degree of flexibility. For example, new attributes and behaviors can be introduced by updating class definitions. That is, there is

10   no need for modifying the "language" used to define call and feature processing logic.

The basis for call and feature implementation in the system of the invention is a run-time object model that exports access to its attributes and behaviors to an external logic definition environment. The run-time object model can include, for example,

15   compiled C++ objects that export their attributes and behaviors to such an environment.

FIGURE 17 is a diagram that depicts an exemplary technique, presented for illustrative purposes, by which a compiled C++ object may export its attributes and behaviors to the logic definition environment. This technique is presented for illustrative

20   purposes.

In C++ typing system, a caller must know an object type in order to call a member function of the object. Therefore, a mechanism is needed for resolving type linkages at run-time given a logic structure defined in a text-based document.

25

The technique illustrated in the diagram utilizes two abstract classes, namely, ActionExporter and Action. An ActionExporter is a class that exports access to one or more of its member functions. An Action is the mechanism for exporting access in a type specific and type safe manner. ActionExporter defines a single, pure virtual

30   member function doAction(Action &). The sole parameter to this function is a

43

reference to an `Action` instance. Since the method is pure virtual, each derived class must provide its own implementation of `doAction`.

`Action` defines two, pure virtual member functions `doAction (Call &)` and
5 `doAction (Feature &)`. The function name is overloaded on parameter type; `doAction` may be called with a reference to a `Call` instance or a reference to a `Feature` instance. This establishes `Call` and `Feature` class instances as the only recognized exporters of behavior via `Action` instances.

10 Two classes are derived from `ActionExporter`: `Call` and `Feature`. Each has three primitive member functions and each provides a similar implementation for `doAction`. A call to `doAction` results in a call to the specified `Action`'s `doAction` function passing a reference to `this`. Conceptually, when a `Call` or `Feature` is instructed to "do an action", the result is "do the specified action using myself." `Call`
15 and `Feature` each have a corresponding derivative of `Action`: `CallAction` and `FeatureAction`. These classes provide a mechanism for catching certain type mismatches that may result from improperly defined logics, for example, a `CallAction` passed to a `Feature` instance and a `FeatureAction` passed to a `Call` instance.

20 FIGURE 18 is a diagram illustrating an example of the class hierarchy derived from `Action`. Classes derived from `CallAction` and `FeatureAction` correlate to the primitive functions to be exported by the `Call` and `Feature` classes. A leaf class in the `Action` hierarchy corresponds to each primitive function exported by the corresponding `Call` or `Feature` class. Each leaf class instance defines `doAction` function to call the
25 corresponding `Call` or `Feature` primitive method. The calling cycle is thus completed. When a leaf class instance is called with a reference to a `Call` or `Feature` instance, it in turns calls the appropriate member function.

The above example demonstrates a technique for exporting access to functions in
30 a compiled object model. A further appreciation of the technique may be attained by reference to FIGURE 36B.

44

As stated previously, the call and feature definition language can represent finite state machines. FIGURE 19 illustrates an exemplary state machine that can determine the parity of a string of 0s and 1s. This machine has three states, Initial, Running, and

5   Done. The machine recognizes only three events; a 1, a 0, and last; last indicates the last data value has been seen.

Two logic definitions, one defining an even parity machine and the other defining an odd parity machine, are presented. Although the logic texts were not defined using a

10   markup language, their structure is hierarchical and could easily correlate to a markup structure. The process for converting these logic descriptions to an executable form includes the following steps:

(a) parsing the text file as specified by the document type definition,

15   (b) verifying that each element corresponds to an appropriate logic element defined by the state machine's object class specification, and

(c) constructing a finite state machine consisting of Action instances corresponding to the behaviors referenced in the logic specification structured according to the specified state and event relationships.

20

A generalized parser, for example, can perform step (a). A "compiler" can perform steps (b) and (c). The result is an object based finite state machine logic that can be executed by a generalized engine. The execution engine operates at the abstract level, i.e., the execution engine operates with `ActionExporters`, `Actions`, and object based

25   finite state machines. Type resolution at run-time is handled, for example, by the virtual method calling capabilities of C++.

FIGURE 20 illustrates a diagram containing an approximate schema for the logics defined previously. The logic consists of a set of *states*. Each state responds to one or

30   more *events*. Each event has one ore more *action sequences*. Each action sequence consists of an optional *predicate*, one or more *actions*, and a corresponding *next state*.

Each action may have one or more corresponding *parameters*. Predicates, actions, and parameters each correspond to an action exported by the run-time object model. A predicate yields true or false, an action is a simple function, and a parameter yields an attribute value.

5

When an event occurs, the corresponding action sequences are identified and processed as follows:

```
for each sequence
do
        if predicate is nil or do predicate is true
        then
                for each action in sequence
                do
                        for each parameter with action
                        do
                                save value of do parameter
                        done
                        do action passing saved values (if present)
                done
                set state to next state
                exit for loop
        endif
done
```

FIGURE 21 illustrates an object model for a finite state machine consistent with the previous description. A state consists of one or more transitions. Each transition consists of one or more associations, called transitions, between an event and one or more action sequences. An action sequence consists of no more than one predicate action and one or more actions. An action may be associated with one or more parameter literals and/or one or more parameter actions. Each action sequence is associated with one state.

An application context controlled by a finite state machine exists in a single state and responds to specified events. An application context consists of one or more application methods (functions). Each such method is exported as a predicate action, an

46

action, or a parameter action. A method is generally exported as a single action type.
Predicate actions yield true or false. Actions are simple functions, and parameter actions
yield generic value wrappers. A value wrapper is a generic holder for attribute values.
Parameter literals represent values embedded within the logic definition.

5

System Components

FIGURE 22 schematically illustrates exemplary TSP/CSX class 4 and class 5
switches according to the invention having several TSP processor nodes and several CSX
10   switch nodes. The aggregated TSP nodes and associated peripherals together constitute
the TSP. Each CSX switch element can be considered individually due to its unique
arrangement of network and subscriber access facilities. Together, the TSP and CSX
nodes are considered a single consolidated system.

15   A exemplary embodiment of the TSP/CSX system of the invention includes at
least two TSP processor nodes, redundant or fault resilient persistent storage devices, and
an isolated LAN dedicated to communication between TSP processor nodes and the CSX
central shelf controllers. The TSP/CSX system can further include an IP based LAN,
independent of the TSP/CSX communications LAN, that supports distributed TSP
20   application and third party software components. The IP based LAN also provides access
to external systems as necessary to support operation of the TSP/CSX within a carrier's
environment.

Each processor node is capable of operating in an independent fashion. Each
25   processor node can include its own independent CPUs, RAM, etc. Power supplies can
be independent unless the supply itself is fault resilient. The processor nodes may be
housed separately or together depending upon the most desirable configuration consistent
with the target footprint, unit cost, and so forth.

30   Each processor may have a dedicated persistent storage device so long as the TSP
application software, possibly in conjunction with third party software components (such

as data management software), provides an acceptable level of protection against loss or destruction of vital system and/or carrier data.

5   Further, the two processor nodes may share a single storage device so long as the device is inherently fault resilient or, in conjunction with other third party software, can otherwise provide an acceptable level of protection against loss or destruction of vital system or carrier data.

10   Each consolidated system provides connectivity between the TSP and at least two central shelf controllers for each CSX node. The LAN is selected to be fault resilient and/or provide redundant pathways between at least two TSP nodes and each central shelf controller pair.

15   The IP based LAN is selected to provide sufficient bandwidth to support distributed software components and communications traffic between the TSP processor nodes and external systems. In addition, additional communications pathways can be added as may be required.

TSP/CSX Communications

20

FIGURE 23 illustrates exemplary communications paths within an embodiment of a consolidated TSP/CSX system of the invention. Each TSP processor node has access to a shared persistent data store. Each node also supports man-machine interfaces and interfaces to authorized external systems such as billing systems, and subscriber

25   provisioning systems. Database access and external interfaces may be supported by a single LAN. Alternatively, separate LANs may be utilized to provide the desired system performance.

A dedicated LAN having a bandwidth, for example, of 100 mbps, interconnects

30   each TSP processor node and CSX central shelf controllers. This LAN provides the communications backbone of the consolidated system, and is the primary means for

accessing the CSX switches. The LAN configuration provides a single communications path between each TSP node and each central shelf controller. The LAN hub is fault resilient. The LAN also supports TSP node-to-node communications for call and feature processing and system and network management. In addition, there is a 10-megabit

5   communications channel between the CSX central shelf controllers. This channel supports CSX internal functions.

As depicted in FIGURE 24, a message distribution and transport (software) layer is interposed between the TSP/CSX application software and the underlying network

10   interfaces. The purposes of this layer include, for example, presenting the TSP as a single logical entity to the CSX application software, and establishing and maintaining reliable point-to-point connections between TSP nodes and CSX central shelf controllers. This layer is statically configured to include provisions for alternate routing in support of fault tolerance.

15

The distribution and transport layer hides details regarding the configuration of TSP nodes from the CSX software, and provides deterministic inter-processor routing of messages associated with specific physical locations. Messages related to specific physical entities, such as endpoints, trunks, are routed to the TSP/CSX elements having

20   those entities within their scope. The distribution and transport layer has access to necessary physical configuration data needed to ensure consistent and reliable message routing between processing elements.

Fault Tolerance

25

Fault tolerance refers to the ability of a system to provide continuous application services in spite of faults and processing interruptions that may occur within the system. These include the ability to continuously satisfy service requests on demand as well as the ability to complete processing of all service requests. For example, with respect to

30   the TSP/CSX of the invention, fault tolerance refers to the ability to complete calls, call features, and call services as well as the ability to complete calls, call features, and call

services once such services have been initiated. These capabilities further include reliable capture and preservation of information utilized for billing, service management, etc.

5   A system that is fault tolerant typically has the following characteristics. It includes, for example, redundant processing elements, and has the ability to dynamically substitute one processor for another. Such a system also provides redundant communications paths between processing elements and the ability to dynamically substitute one communications path for another. Further, a fault tolerant system typically

10   provides redundant system access paths between the system and its client community, and the ability to utilize any available path to provide service. In addition, the system can have redundant or multiple persistent storage elements that provide continuous access to information used by application services, and that provide reliable persistent storage of information produced by application services. Redundant data access paths including the

15   ability to substitute one data access path for another are also provided.

          In a fault tolerance system, one processing element may be called upon to substitute for another under the following exemplary circumstances:

20   (a) a processing element experiences a processing fault or failure that prevents it from handling new service requests and/or prevents it from completing one or more services already in progress,

          (b) a processing element is administratively prohibited from handling new service

25   requests and completing services already in progress,

          (c) a processor is administratively prohibited from handling new service requests but is permitted to complete services already in progress.

30   In case (a), the system recognizes that a fault or failure has occurred, determines that the affected processing element cannot or should not continue processing, and shifts

50

responsibility to one or more alternate processing elements. In cases (b) and (c), an external authority has issued a command to shift processing responsibility from one processing element to one or more alternative processing elements. The above case (b), sometimes called a *forced switchover*, is typically a response to an event or set of events

5 within the system that call for urgent re-action to prevent loss of service. The above case (c), sometimes called a *graceful switchover*, is typically a pro-active measure to prevent service degradation or to initiate some system maintenance activity.

The decision-making and failover execution mechanisms need not be integrated.

10 A single switchover capability that can serve in forced and graceful situations can be activated by any number of decision-making capabilities including internal and external decision sources.

One communications path may be substituted for another under the following

15 exemplary circumstances:

(a) a communications path has failed or become un-reliable, and/or

(b) a communications path is administratively removed from service.

20

The determination that a communications path has become unreliable is generally carried out by a processing element within the system. However, the decision to substitute one communications path for another due to poor reliability can be made by an external entity.

25

Selection and control of data access paths is normally confined within the system itself. In particular, decisions regarding the viability and selection of a data access path are usually handled exclusively by the system. Redundancy of storage elements is usually controlled by the data management system. The data management system may offer the

30 ability to configure storage elements but usually does not provide run time control.

51

A system that can continue and complete at least some services in progress and/or satisfy at least some new service requests is referred to herein as being partially fault tolerant. A system that can continue and complete all outstanding service requests and satisfy all new service requests is herein referred to an being completely fault tolerant.

5

The system of the invention provides at least partial fault tolerance of single points of failure. That is, the system can continue operation even if the failure of a single system component occurs. For example, the system of the invention provides continuous support for the initiation of calls, call features, and call services. It further provides

10 continuous support for stable calls, call features, and call services. For telephony services, calls wherein both parties have engaged in two-way conversation are traditionally considered stable. In the system of the invention, a call, call feature, or call service that is in a stable state, is not adversely affected by the failure of a single system component. In addition, the system of the invention prevents the loss of billing data

15 related to completed calls, call features, and call services. It further reduces the loss of system data such as logs, performance statistics, usage statistics, etc. Those skilled in the art will appreciate that the system of the invention can also be configured to provide complete fault tolerance.

20 The system of the invention can declare processor faults upon various conditions that include, but are not limited to, loss of communications between TSP and CSX, loss of access to database, system (internal) resource exhaustion, program exceptions, and audit exceptions. A processor may declare its own failure or be declared failed by an external entity. In some embodiments that do not provide communications path

25 redundancy, a variety of problems including loss of a physical connection, system level software errors, and a non-responsive far end processor (either TSP node or CSX switch controller) appear as loss of communications between two elements. Loss of a single physical connection or a software exception associated with a single connection appears as isolation of the connected processor, e.g., loss of connection between a TSP node and

30 the LAN hub results in CSX switch controllers talking to, for example, one TSP node.

Apparent isolation of a TSP node mandates a TSP failover, and apparent isolation of a CSX switch controller mandates a controller failover.

5   In some embodiments, a variety of problems including disk controller failures and system level software errors appear as loss of communications between a TSP and the database. Loss of database access mandates a TSP failover.

Further, to the extent that system level resources can be identified and quantified, the system can monitor resource availability and declare faults upon resource shortages.
10   In addition, to the extent that program exceptions can be identified and "caught", the system can recover and declare a software fault. Audit procedures may also be defined to examine various system characteristics and declare faults when acceptable limits are exceeded and the problem can not be corrected.

15   In some cases, for example, in cases of non-responsiveness or unacceptable performance, one processor may declare that another has failed. In one model, referred to an "bee watcher" approach, one processor can monitor another and declare the failure of the monitored processor if it fails to respond to a query, or fails to meet a pre-arranged schedule commitment. In another model, referred to as "breach of service contract"
20   model, a client processor may declare that a server processor has failed when it no longer receives acceptable service (the server may be non-responsive or may respond too slowly). Upon failover, the designated standby or alternate processor must assume the failed processor's role. That is, it must load the applicable context information from the persistent store, prepare to continue viable calls, call services, and call features, or
25   terminate unviable calls, call features, and call services. Loading contexts is necessary because there may be recoverable contexts in states that require termination and for which no external events may be expected.

The process of recovering contexts is preferably governed by a policy that
30   recovers the most useful contexts first, and prioritizes response to external events, representing immediate service need, over recovery. The policy should also prevent a

load spike that would degrade the processor's performance (recovery process should spread processor demand over time to prevent the recovery process from degrading system response to external events). The system has also the ability to recover a context from the persistent store upon an external event. The failover and recovery procedure

5   also ensure that the failed processor does not inhibit service as it is restored to service.

In the system of the invention, one processor can substitute for another to continue a stable call, call feature, or call service. In particular, the system of the invention provides descriptions of calls, call features, and call services, which include the

10   notion of condition or state, that are transportable from one processor to another. As the detail and accuracy of these descriptions increase, the system's level of fault tolerance also increases. For example, if a call is waiting for the result of a database transaction, that information will be lost following a processor substitution unless the pending transaction is represented in the call's description. If the call description includes a

15   description of the pending transaction, the transaction request could be re-issued as a recovery measure upon failover.

Redundancy Configurations

20   The architecture of the system of the invention offers a variety of redundancy configurations. The TSP/CSX system of the invention can include (a) redundant processing nodes within the TSP, (b) redundant switch controllers within the CSX, (c) fault tolerant object database shared by the TSP nodes, (d) fault resilient 100 Mbit LAN hub interconnecting the TSP processing nodes with the CSX switch controllers, and (e)

25   separate man-machine and external system interfaces for each TSP processing node.

Redundant TSP processing nodes provide the basis for implementing fault tolerant application software. The processing nodes are independent, and fault tolerance features are provided by the application software. In one embodiment, two TSP

30   processing nodes share a fault resilient object database. This database provides persistent storage for all system data and is the mechanism for sharing run time state information

between the two TSP nodes. Further, a fault resilient LAN Hub provides communications paths between the TSP nodes and the switch controllers. One embodiment includes the following four distinct communications paths:

5   (1) Node 0 to controller 0, (2) Node 0 to controller 1, (3) Node 1 to controller 0, and (4) Node 1 to controller 1. Each path consists of two connections, namely, (1) Node to Hub, and (2) Controller to Hub. This configuration can tolerate loss of a single node to hub connection, of a single controller to hub connection, or loss of one node to hub connection and one controller to hub connection. Loss of a TSP node to hub connection

10   prohibits load sharing between the TSPs and may or may not mandate a TSP node failover (a switch controller failover would not be necessary). Loss of a switch controller to hub connection does not prohibit TSP load sharing and does not require a TSP node failover, but it may require a switch controller failover. Loss of two connections prohibits load sharing and may require either a node failover or a switch controller

15   failover.   Each TSP can support one or more connections to external systems and/or machine interfaces.   A TSP node may or may not have redundant communications paths to these external entities.

       FIGURE 25 depicts an $N + 1$ configuration consisting of four TSP Nodes and

20   three CSX switch elements. Each of the three primary TSP nodes is dedicated to a separate CSX switch element. The standby TSP node is configured to support any of the three CSX elements. Hence, the standby node may substitute for any of the primary nodes. However, the standby may substitute for only a single primary node at any given time. The system can compensate for the loss of a single primary TSP node by

25   substituting the standby processor in its place. Further, a $1 + 1$ redundancy can be achieved by providing one standby TSP node for each primary node and configuring the communications interfaces accordingly.

       FIGURE 26 shows another configuration in which the traffic load for each CSX

30   has been partitioned into thirds and distributed equally among three primary TSP nodes. If one of the primary TSP nodes fails, each CSX utilizes the communication layer to map

the failed node's partition to the standby node. Similar to the previous configuration, the system can substitute the standby node for any one of the primary nodes if a primary node fail.

5   In the configurations shown in FIGURE 25 and FIGURE 26, each TSP node is supported by a shared object database. This database provides the standby with access to the objects needed to continue processing including access to transient call objects. In effect, the shared transient data store is the mechanism for "copying" objects from one processor to another.

10

Application Software Redundancy

Application software can employ transportable context objects to capture and maintain the state of application services. A context object represents or describes the

15   state of an application service such that an arbitrary processor can continue and complete the service based on the context object. Context objects may be aggregated in order to represent the state of a complex application service.

A processor needs to have, or to have access to, context information, describing the

20   state of each application service that is in progress, in order to substitute for another processor. Hence, the system provides some transportable mechanism of describing the state of application services.

A task (thread or process) is a mechanism that encapsulates processing. A task

25   carries with it the ability to "run" an application. Assuming that application contexts can be transported from one processor to another, it must also be possible to associate a task with each context to facilitate execution. The ability to dynamically bind tasks with application contexts is therefore important to fault tolerance. Dynamic task-context bindings also discourage the capture of essential context information within the execution

30   call stack.

The application software provides some mechanism for rendering contexts present on or accessible by the alternate processor when a processor substitution occurs. For example, replicating contexts in an alternate processor, storing contexts on a sharable disk, will ensure that a substitute processor can continue application services in progress

5   following a processor substitution. Further, event-context binding is rendered deterministic and consistent across all processors. For a particular event, each processor must be able to bind the event to a unique (and correct) context, and to deliver the event to the appropriate task-context association for execution. This includes consistent recognition of events that call for the creation of new contexts.

10   Event-processing mechanisms are also defined that produce consistent behavior across multiple processors. This is often inherent in distributed software—each processor runs the same application software. However, to the extent that applications incorporate dynamic elements, mechanisms are provided that ensure consistent application behavior across multiple processors. For example, a processor that substitutes for another

15   processor must have access to all applicable CPML pages in order to ensure consistent behavior.

Further, blocking system and infrastructure services (functions, method calls, etc.) can be converted into asynchronous client-server services. System and infrastructure

20   services may be synchronous if and only if they are guaranteed to be non-blocking. For instance, reading the system clock is typically non-blocking and can be implemented and accessed synchronously. However, reading an object from a database is typically blocking and hence is modeled as an asynchronous client-server service. This prevents the capture of context information within the execution stack—a form of context

25   information that cannot be transported from one processor to another. For example, the fact that an application service has a database transaction pending is context information that should be represented explicitly in a context object rather than implicitly by a task that is blocked on a function call. It further prevents a temporary static binding between a task and a context object. Static task-context bindings should be limited to

30   infrastructure level services because this context information would be lost in a processor

57

switch. Database requests should be represented within application contexts if such requests are to be preserved across processor failovers.

5   The ability to vary task to context ratios, dynamically if possible, can improve overall performance with fewer tasks. This means that the ratio of tasks to context objects may exceed one to one. If a task becomes statically bound to a context, even temporarily, it can impede overall throughput across its associated contexts.

10   Synchronous interfaces are permitted between objects only under the following conditions:

(a) Guarantee of locality—a client object must be guaranteed the server object is local and can be referenced directly,

15   (b) Guarantee the service is non-blocking—an application level service that is blocking must be modeled as an asynchronous client-server interaction for the reasons cited previously,

(c) Guarantee the service is constant with respect to the server or that suitable

20   contention management mechanisms are in place,

(d) Guarantee the service is public or that the client has the privileges needed to access the service.

25   Fault and failure recognition and response mechanisms are defined such that processor substitutions occur, as needed. The system of the invention is capable of identifying conditions that represent or may represent an operational interruption. It is also be capable of responding in a way that prevents or minimizes loss of service, consistent with the system's requirements.

30

In addition, a system management capability is defined such that processor substitutions can be commanded by an external entity. Loss of external events, messages, and commands are prevented. Moreover, mechanisms are defined for altering external event, message, and command routing schemes when processor substitutions occur.

5

Application and certain infrastructure components are implemented using finite state machines (FSM) to control behavior. This is a natural fit to the application, a commonly understood implementation model, and lends itself to clear definition of state and context and separation of state/context from processing.

10

Well-defined FSMs are context dependent but also context free—a FSM instance contains no context data. It describes or defines behavior in terms of a *context* (object or set of objects) and the state of the context. This encourages well-defined contexts that provide a high fidelity representation of application state that can be transported from one processor to another. Moreover, FSMs can be implemented as singletons (single instance within a processor) and placed in the global scope with minimal consumption of memory. Alternatively, FSMs can be defined by data structures (e.g., XML document), placed in persistent storage, and loaded into memory on demand. In this case, object caching may be used to reduce the impacts of persistent storage I/O.

20

It is also possible to define a stack execution model based on FSMs. According to this model, one FSM can invoke another using a stack mechanism and a standardized data-passing interface. This model supports the dynamic integration of FSMs defined by different domains within a common application context. It extends the expressive capability server object interfaces and provides a useful layer of encapsulation for the client object designer. For example, an application or infrastructure server object could extend its interface definition to include a FSM that is "invoked" at run-time. This approach can simplify the interface between objects and can hide the underlying details of the interaction between client and server.

30

An interface designed and implemented as an FSM can encapsulate many object interaction details including the locations of client and server, the message objects exchanged between client and server, the message exchange mechanism, the service's mode of execution (synchronous or asynchronous), error detection and handling

5   (including potential recovery options), etc. This approach can simplify application development.

The infrastructure of the system of the invention provides an object class, e.g., Task, to encapsulate OS threads. Application objects that require processing support

10   request it from the infrastructure instead of creating new threads at will. This relieves the application from dealing with the intricacies of thread invocation and control, and it provides a consistent approach to multi-threading throughout the application. Furthermore, it provides a point of control that may be used to manage the number of threads and the way the threads are aligned to various task categories within the system.

15   Adjusting the ratio of tasks to task categories can allow controlling priorities without manipulating OS visible thread priorities. For example, adjusting the ratio of call processing related tasks to the number of maintenance related tasks can effectively raise or lower the priority of call processing.

20   The infrastructure also provides support for creation and destruction of task objects, associating and disassociating tasks and application objects on demand, and publicizing these associations to facilitate message (event) delivery. This support may also include features such as task categorization in support of priority management, task to application object ratio control for load balancing, etc.

25

The mechanism for exchanging data between objects can be integrated with the overall execution model. An object class, e.g., Event, provides the basic vehicle for exchanging information between objects. This class is specialized (sub classed) to provide application and infrastructure specific event classes.

30

60

The Event class defines two points of interaction between the infrastructure and application domains:

    (a) Event routing—this is the process of determining which processor should
5   handle a particular event. This behavior is application defined and is therefore represented as an abstract method.

    (b) Event dispatching—this is the process of determining which object, within a processor's scope, should handle a particular event. This behavior is also application
10   defined and is represented as an abstract method. Event dispatching is preferably non-blocking. Assuming that the dispatching behavior is non-blocking, the event can be immediately dispatched to the appropriate application object. If the dispatching behavior is blocking, this behavior must identify an application domain object to carry out that behavior and finish dispatching the event.
15

    The infrastructure can invoke each of these methods as needed when transporting events between objects. Application class events are routed to a processor and then dispatched to an object—this removes assumptions about where an event can or will be processed. Infrastructure class events may not require routing because infrastructure
20   services are available globally. Infrastructure support shall be provided for the exchange of events with external systems, routing of events received from external systems, and event dispatching.

    Delegating event dispatching to the application provides flexibility and a
25   mechanism for solving boundary conditions, e.g., context object creation. If a particular event requires a new context object, the application can define a dispatching algorithm that targets these events to an object factory, or it can construct a context object that it then specifies as the destination. Object factory is a pattern wherein an object is responsible for constructing instances of an associated class.
30

A secondary goal of the overall task-processing model is to reduce contention across highly dynamic objects. Call processing objects are expected to frequently dynamically change state and context. To simplify contention management, each such object is supported by a single task. All events related to a particular call-processing object are routed to its associated task. This reduces the possibility of contention for a single object by multiple execution threads. This also yields an event model in which events always appear in a serialized fashion—even if two or more events actually occur simultaneously.

A shared, fault tolerant persistent storage device houses context objects, i.e., event-processing objects, data objects used to rout e events to context objects, and system data objects.

Further, data access mechanisms and processes are designed to access the shared data store dynamically as needed to fulfill an application's request for data. That is, if a target object cannot be located in memory, it will be accessed on the shared store. Only upon failure to locate a target object on the shared store does the application determine that the target object does not exist and/or declare a fault.

Caching, i.e., application level caching, is employed to optimize access to data structures and objects that are used frequently. Initial attempts to search or retrieve any data object is applied to cache. However, all access mechanisms are designed to access the persistent store in the event of a cache miss.

Data organization structures, e.g., hash tables and indices, are also subject to dynamic access mechanisms. Accessing an object does not fail unless the associated organizational structures have been destroyed or have become corrupt.

Dynamic access mechanisms simplify certain aspects of failure recovery since there is no fixed expectation that a particular object will be in memory when it is accessed. However, care is taken to ensure that nominal access modes are not severely

impacted by dynamic access mechanisms (some balance between dynamic access and static data loading may be needed). However, providing a dynamic access mechanism for all data objects offers the maximum flexibility for defining data storage policies.

5   The infrastructure also defines a framework for routing and dispatching events. For certain event classes, e.g., event classes related to call processing and other services related event, these decision-making algorithms are defined by the application. For other event classes, e.g., event classes related to infrastructure services, these decisions are defined in accordance with architecture/infrastructure policy.

10

The infrastructure accesses routing and dispatching behavior at appropriate points in an event's transmission cycle. Ensuring appropriate and consistent results are the applicable domain's responsibility—application or infrastructure.

15   For static data and program elements, i.e., data and program elements that are static for the life of the application, ordinary distributed software practice will suffice. For dynamic elements, including program and data elements as well as sharable contexts, the architecture provides mechanisms for maintaining consistency across multiple processors.

20

Dynamic data and program elements can be categorized as follows:

(a) elements for which the persistent store is the database of record and whose copies may be contained in volatile memory stores across multiple processors. An
25   external business or operations process generally controls these entities (the authority to establish and modify the state of these entities is outside the scope of call, call feature, and call service processing),

(b) elements for which a volatile memory store is the database of record and
30   whose copies are placed on the persistent store in support of application features such as

fault tolerance. Call, call feature, and call service processes generally control the state of these entities.

The integrity of the entities having persistent store as database of record is
5   ensured by propagating state changes from the persistent image to memory resident copies. Memory resident copies of these entities are discarded once they are no longer in use.

Each time an entity is loaded into a volatile memory store, a dependency is
10   established between the source entity and the copy of that entity. Modifications to the state of a source entity are ultimately reflected by each of its memory resident copies, or dependents. Explicit representation of a source entity's dependencies provides a mechanism for identifying and locating its dependents.

15   The nature of a source entity can suggest a priority for updating its dependents. Certain entities may require immediate update of each dependent. For instance, changes in operational policy may require immediate update of all memory resident copies of that policy. Other entities may tolerate some delay in updates to dependents. For example, modification of a group level feature may require immediate application to all new
20   instances of that feature while permitting instances in progress to conclude under the previous policy. In essence, updates to the feature's dependents are not required until a new feature instance is initiated.

Immediate update of a memory resident entity introduces contention—one or
25   more application objects may hold a reference to and be accessing the entity in question at the time the update must occur. To ensure integrity, some form of mutual exclusion lock must guard access to each entity requiring immediate propagation of updates across all copies.

30   Delayed update of memory resident entities can be handled more simply. Upon a source entity's modification, its dependents are located and "privatized." Application

objects possessing references to an outdated dependent entity may continue to use that entity as long as the reference is held. Once all references to the outdated dependent are released, it is discarded. However, establishing a new reference to the modified entity will cause a new copy to be loaded from the persistent store.

5

Integrity for entities having volatile store as database of record is ensured by propagating state changes from the volatile memory image to the persistent image. Persistent copies of these entities are discarded once they are no longer in use. The application controls these entities and determines the circumstances for updating the

10   persistent image.

The infrastructure defines an asynchronous, message-based capability for communications between objects. This is a natural basis for a message-based, client-server model for system and infrastructure services. The model is already required for

15   the application domain. Hence, it is logical to extend it to the infrastructure domain. This also presents one consistent activa tion mechanism whether a service provider exists in the application domain or the infrastructure domain.

It is useful to provide a consistent activation mechanism for both synchronous and

20   asynchronous services, i.e., it is useful to encapsulate whether or not an infrastructure service operates in a synchronous or asynchronous fashion. This further simplifies the application developer's role and provides additional flexibility. It is also useful to define infrastructure services that dynamically choose an asynchronous or synchronous mode depending upon the circumstances. For example, a data access service could operate

25   synchronously if the requested object is contained in the memory cache or asynchronously if a transaction must be issued to retrieve the requested object—either way the presentation to the application is identical and the only variance is performance. However, it may be useful to define synchronous interfaces for infrastructure services that are guaranteed to be locally available and non-blocking.

30

Class 4/5 Configuration

FIGURE 27 provides an overview of an exemplary installation of a TSP/CSX
consolidated system of the invention. The system includes two TSP nodes with
appropriate interfaces to the carrier's external systems such as the operational support
5   system (OSS), the subscriber provisioning and maintenance system, the billing system,
etc. As shown, a single LAN supports these interfaces. Additional interfaces may be
used if needed to provide sufficient throughput on these external interfaces.

The TSP is supporting a single CSX, and the CSX has two central shelf
10   controllers appearing on the communications LAN between the TSP nodes and the CSX.
Digital subscriber loops and DS1 access facilities provide subscriber access. The DS1
access facilities are presented to the CSX by remote data terminals using the GR303
protocol. The CSX also provides access to the PSTN via DS1 network access facilities.
The network access facilities may support a variety of protocols including, for example,
15   FGD, ISDN/PRI, and SS7/ISUP.

The carrier offers features through three levels of service subscription. In one
level of service, a subscriber has access to features to which the subscriber directly
subscribes at an individual level. In a second level of service, a subscriber is a member
20   of a subscriber group that has subscribed to a set of features. Each member of the group
has access to the group's associated features. In a third level of service, a subscriber has
access to feature offered, by default, to all subscribers by the carrier.

Application Framework
25

FIGURES 28A and 28B depict a basic application framework that is utilized in
one embodiment of the invention. An alternative to the foregoing is the use of CORBA
Orb's to effect interprocess message handling.

30   Messages are received via an external interface component and converted into
events as specified by a message protocol. Events are dispatched to the appropriate

application context for processing. Inter-object communications, including inter- and intra-processor communications are also supported by the event passing capability. Blocking infrastructure services are implemented in the same fashion as application services using a similar task-processing context binding.

5

There are two stages to event passing, i.e., event routing and event dispatching. Event routing is a process that identifies the processor that houses the appropriate processing context for an event. Event dispatching is a process that identifies the application context, within a processor, that is responsible for processing an event. The

10　　appropriate application domains define the event routing and event dispatching processes (the infrastructure merely executes these behaviors at the appropriate point in the event passing sequence).

Events received from external systems are dispatched without routing. It is

15　　assumed that such events have been routed correctly. Events issued from an infrastructure service context are also dispatched without routing. It is expected that infrastructure services are ubiquitous and therefore that inter-processor event passing is not required to access these services.

20　**External Interface Object Model**

With reference to FIGURE 29, the TSP can exchange messages with an external system, such as the CSX. An interface characterizes the messaging association between the TSP and the CSX. The interface sends and receives message fragments. One or more

25　　message fragments constitute a message. Messages are converted to events and events are converted to messages according to a message protocol.

A message protocol consists of a parser and a streamer. A parser converts a message to an event. A streamer converts an event into the corresponding message. This

30　　arrangement completely encapsulates the mapping of messages to events and events to messages. New external interfaces may be introduced without modifying the event

67

hierarchy so long as the new protocol corresponds to existing events. Introduction of a new protocol may introduce new events, as well.

Event Passing

5

FIGURE 30A depicts an event trace outlining the process of receiving a message, converting the message into an event, and dispatching the event to the appropriate processing context in a system of the invention.

10 FIGURE 30B presents a diagram illustrating an event trace that outlines the process of routing an event and sending it to an external processor via an external interface, including conversion from an event to a message format.

FIGURE 31 is a diagram illustrating the nature of an event hierarchy. Such a
15 hierarchy provides considerable flexibility in routing and dispatching. For instance, all TCAP events could be dispatched to a single processing context that handles all TCAP messages by implementing the routing method in the TCAP class. Alternatively, each event class derived from TCAP event could be routed to an application context dedicated to handling events of a specific type, for example, 800 Number Translation, Calling
20 Name Delivery, and Local Number Portability, by implementing a unique dispatch method for each class.

Data sets

25 The data sets that are associated with the TSP (either supporting the TSP or being generated by the TSP) fall into at least five categories. These categories include, for example, (a) data sets that are accessed frequently but whose (persistent) values change infrequently, (b) data sets that are accessed frequently but whose (persistent) values change with moderate frequency, (c) data sets that are accessed frequently and whose
30 (persistent) values change frequently, (d) data sets that are accessed with moderate

frequency and whose values change frequently, and (e) static data sets that are created frequently and accessed only infrequently.

Route and Call/Feature policy data sets belong to the above category (a). A summary of these data sets are presented in Tables 34 and 36 of FIGURES 32A and 32B. These data sets support processing of each call, and their database of record in the persistent store. The memory capacity of the system of the invention is preferably sufficient to support complete caching of these data sets. The use of these data sets allows achieving high memory residency, and maintaining consistency with the database of record when updates occur.

The above category (b) of data sets includes primarily the endpoint/subscriber data set, a summary of which is presented in Table 38 of FIGURE 32C. This data set supports processing of each call. Although the endpoint/subscriber data values change with greater frequency than those of the Route and Call/Feature policy data sets, they change at a significantly lower frequency than call attempt rates. The database of record for this data set is the persistent store. The endpoint/subscriber data set allows achieving relatively high memory residency with reasonable memory consumption, and maintaining consistency with the database of record when updates occur. The data corresponding to frequency callers achieves higher memory residency than those corresponding to infrequent callers. Further, the memory capacity is preferably sufficient to support complete caching of this data set for a mature, fully configured system, thus achieving good disk access performance upon cache misses.

The above category (c) data set includes primarily call and feature state data, of summary of which is presented in FIGURE 32D. This data set supports processing of each call and its value changes frequently during call and feature processing. The frequency of changes in persistent value is less than overall frequency of state change, but it is at least several times per call. The database of record is the memory image. A persistent image is maintained for selected states to support fault tolerance. This data set allows achieving complete memory residency, maintaining consistency between the

memory and persistent images for selected states, thereby providing excellent disk access performance for all consistency updates.

5   The above category (d) consists primarily of equipment and transmission facility state and statistical data sets. FIGURES 32E, 32F, and 32G provide summaries of these data sets. The data sets support network management and system operations. Some members of this data set can change their values as a function of call volume and may be subjected to frequent value changes. Aggregate members are also subject to frequent value changes. Routine polling intervals can govern access to this data set. The database of record is a memory image. Persistent images are provided for historical statistical records (usage and performance) only. The data set allows achieving high memory residency, and further achieving reliable capture of historical statistical records.

15   The above category (e) data set consists primarily of Call Detail Record (CDR) and Automated Message Accounting (AMA) record sets. FIGURES 32H and 32I provide summaries of this data set. CDR's are generated for each call and/or feature. In selected number of cases, multiple CDR's may be generated for a single call. Once generated, CDR's are captured to disk for subsequent processing. AMA records are in turn derived from CDR's. The database of record is the persistent image. CDR memory
20   images exist only as needed to support reliable capture to the persistent store. That is, there is no memory image of AMA records. This data set allows achieving reliable capture of CDRs to the persistent store. Much of this data set is also subject to sharing and duplication. Sharing is utilized to make the data available to multiple processing elements. Replication is utilized to support fault tolerance.
25

Data sets that are related to physical equipment and facilities, and related to compiled software elements are not subject to frequent schema changes. Other data sets may experience occasional schema changes. In particular, some embodiments of the invention provide schema flexibility for log data types, such as system logs, exception
30   logs, audit logs, CDRs and AMA records. This can allow altering the contents of these data sets over time without software compiles.

The persistent storage subsystem of the system of the invention provides data sharing. That is, data can be shared among multiple TSP processing nodes. This includes data housed by the TSP on behalf of CSX switch elements, thus ensuring high

5   availability of data elements required by the CSX. Further, the persistent store provides mechanisms for controlling object memory residency (cache management). The purpose of this capability is to provide access performance sufficient to meet the systems's overall performance objectives. In addition, the persistent store allows processors to share transient call/feature state and context objects through data replication. The persistent

10   store also supports multiple threads per process and multiple processes per database. Further, the persistent store provides object level control of database commits to ensure that transient objects are persisted in correct (e.g., stable) states. Another functionality of the persistent store is object level locking. This allows achieving the overall performance objectives by reducing contention to the object level, thus decreasing the occurrence of

15   multiple threads accessing the same object. In addition, the persistent store supports evolution of the object while the system is running.

FIGURE 33 illustrates a diagram depicting the nature of the correlation between call/feature states in the volatile store and in the persistent store. The states in the

20   diagram are arbitrary and only for illustrative purposes (they do not correlate to any real call model). When a call is created, the initial state is persisted to the transient data store. This facilitates recovery processing by representing resources that have been in use, but for which no stable call state was ever reached. As the call progresses, it ultimately achieves a stable state. At this point, the image in the persistent store is updated to reflect

25   the stable state. As the call progresses further, it achieves another stable state causing another update to the transient data store. As the call concludes, the persistent image is destroyed because its only purpose is to support recovery of stable state calls. In the event of a fail over, a back up processor has access to the transient data store, and can therefore recover the appropriate stable call images. The call will revert to the state

30   represented in the transient call store regardless of what the actual call state was at the point of failure.

Additional Embodiment

FIGURE 36 schematically illustrates an embodiment of a system 100 of the invention that converts a textual description in a mark-up language of a process, e.g., a state machine, to an object graph, and preferably stores the object graph in an object repository, e.g., an object database. The object graph can be accessed to determine transition of a present state of the process into a new state and to determine actions, if any, to be performed. As described in more detail below, the system of the invention also provides two stages of validation. In the first stage, a lexical structure of the textural description of the process logic is validated by comparison with a document type description, and a "logical" document object model is generated from the textual description. In the second stage, actions or operations referenced by that document object model are validated, e.g., by comparison with a "reference" object model.

More particularly, the exemplary system 100 includes a parser 102 that receives a textual description of a process 104, modeled as a state machine, in a mark-up language and generates an document object model 106. The parser 102 can be of any commercially available type suitable for such purpose. The textual description of the state machine, herein referred to as the state machine document, identifies one or more states and one or more events that cause transitions between these states. For example, in the telecommunications domain, the textual description can identify a state machine representative of a call feature, e.g., call forwarding, provided to a subscriber. In such a case, the call feature state machine transitions from one state to another in response to one or more events, e.g., call progress events such as hookflash, and optionally performs one or more actions, e.g., routing a call.

The mark-up language can be chosen to be any language that is suited to a particular application. HTML, XML, and other mark-up languages which are extensions of XML (e.g., CPML) can be utilized in a system of the invention. As an illustrative example, FIGURE 38 depicts an XML document that describes a logic for a state

machine that provides a call forwarding feature to a subscriber of a telecommunications service.

The model 106, herein referred to as document object model (DOM), includes
5   objects and/or references to objects that represent the states identified in the state machine document. These objects can also reference one or more operations for performing the actions, if any, specified for various states of the state machine. For example, in a C++ object-oriented environment, the object model can include, e.g., structures (struct) and/or classes that represent the states identified in the textual
10   description. In a telecommunications application, the DOM is an object representation of a call processing logic, e.g., the logic for providing a call feature such as call waiting.

The parser 102 is preferably a validating parser that compares the lexical structure, e.g., syntax, of the textual description 104, i.e., the state machine document,
15   with a reference lexical structure 108, herein referred to as logic document type definition (DTD), to ensure that the structure of the textual description 104 conforms with a pre-defined structure. For example, this validation step ensures that the structure of the textual description that defines a state machine is consistent with a pre-defined structure for defining state machines. Appendix F provides an example of a logic document type
20   definition which provides a reference lexical structure for defining a state machine.

The exemplary system 100 further includes a pre-processor 110 that receives the DOM 106 and generates an object graph 112 which is representative of the state machine defined in the textual description, ie., state machine document. By way of example,
25   FIGURE 36A presents a graphical representation of such an object graph based on logic specifications provided in a mark-up language. In addition to generating the object graph, the pre-processor 110 validates the document object model 106 by comparing it with a reference document object model 114, herein referred to as reference DOM. The reference DOM includes objects which are representative of permissible operations
30   (actions). That is, the reference DOM identifies those operations that the system can support. For example, in a telecommunications application, the object model 110 may

73

have a reference to an operation (action), such as playtone. Validating the object model ensures that such operations are in fact supported by the system. Hence, the object graph 112, generated by the pre-processor 110, provide object representations of validated actions for a given state machine as originally specified in the textual description.

5

The reference DOM 114 can be generated, for example by the parser 102, from a textual description 116 in a mark-up language of the reference model. In addition, the lexical structure of the textual description of the reference model can be validated by comparing it with a reference lexical structure 118, herein referred to as class and

10   instance document type definition.

The object graph 112 includes nodes which are indicative of the states of the state machine, and edges which are indicative of transitions between these states. The object graph 112 is stored in an object repository 120, e.g., an object database. One

15   embodiment of the invention employs an object database sold by Excelon Corporation. It should be understood that other object databases can also be utilized in a system of the invention. As will be discussed in more detail below, the object graphs can be accessed to execute particular iterations of the state machines represented by the object graphs.

20   In some cases, it may be necessary to adapt the object graph 112 for storage into the object repository 120. Hence, the exemplary system 100 includes a database adapter 122 that can transform the object graph, if needed, in a manner that renders it compatible for storage into the object repository 120.

25   Each object graph stored in the object repository generically represents the logic for a particular state machine. This representation is context free in that the object graph does not have any information regarding a particular state of and/or events received from an instantiated state machine, i.e., a particular iteration of the state machine. To execute a state machine, the system of the invention can instantiate an object having information

30   regarding the context of the state machine, i.e., present state of the state machine and events received by the state machine. This context object communicates with an engine

that in turn communicates with the object repository to accesses the object graph corresponding to the state machine. The engine navigates the object graph and instructs the context object regarding the state transitions and/or actions that the present state of the state machine requires. Further understanding of the run-time environment of a

5   system of the invention can be obtained by reference to FIGURE 13 above.

The use of an object graph for execution of a state machine can be perhaps better understood by reference to FIGURE 37 which schematically illustrates the execution of a call feature provided by a telecommunications system of the invention which utilizes an

10   object graph to represent a state machine corresponding to that call feature. A subscriber's phone 122 can interact with switching and access equipment 124, e.g., PBX, to create a connection with a switch process 126.

The descriptio n of the call features of the subscriber are stored as objects in the

15   object repository 120 (FIGURE 36). These subscriber objects can be generated, for example, from a textual description of these features in a mark-up language, such as XML. A subscriber context object is instantiated that monitors progress of the subscriber's calls, and can transmit call events to a feature object instantiated in a feature context to provide particular call features to the subscriber. The feature object

20   communicates with an engine which in turn navigates an object graph corresponding to the call feature, which is stored in the object repository 120, and instructs the feature object the state transition and the actions, if any, that need to be performed in response to a received call event.

25   A system according to the invention provides a number of advantages. For example, defining state machines in a mark-up language provides a more generic implementation of state machine than code based counter parts. Further, such an approach results in an event driven system that is more flexible than code based systems. For example, a system of the invention can be more easily provisioned and implemented.

30   Further, providing an object representation of the state machine results in faster processing compared with interpretive techniques.

Further aspects of the construction and operation of the illustrated embodiment, in particular, for example, system provisioning, feature classifications and interactions, and multi-party calling, can be better understood by referring to the following discussion.

5

Service Provisioning

Call features and services are provided by a set of collaborating feature components. In many cases it is necessary to provision several feature components in

10   order to provision a single call service. In some cases, simply replacing an existing feature component with another provides a new feature. A service may require a collection of specific components or it may require one or more specific components, one or more of which are chosen from a particular feature classification.

15   There are well-defined inter-dependencies between the feature component classifications that describe how components of different classification relate to and interact with one another. These inter-dependencies can be used to support decision making when configuring a particular service.

20   Each feature component is classified and assigned an identifier. From the EMS perspective, feature component identifiers are strings.

The feature classifications and inter-dependencies as well as a basic POTS service configuration are described in more detail below.

Feature Classifications

Origination Request Handler

5   With reference to FIGURE 39, an origination request handler responds to a subscriber's attempt to place a call, invoke a vertical service feature, or invoke a special service. Origination request handlers typically provide feedback to the subscriber, such as dial tone, and gather a DTMF digit sequence that indicates the subscriber's intent. However, origination request handlers may provide unique behavior such as originating a

10   call automatically, providing customized responses, etc.

An origination request handler is usually invoked by a subscriber lifting the telephone handset. However, an origination request handler may be invoked directly by other feature components. For instance, a feature that terminates by "re-originating" may

15   invoke an origination request handler directly.

Each subscriber is provisioned, by definition, with one origination request handler. FIGURE 39 shows that the following two origination request handlers have been defined:

20

1. POTS Origination Request Handler—this feature provides the standard "black telephone" response when a subscriber lifts the handset. This includes providing dial tone, collecting a dialed number, analyzing the dialed number, and invoking the subscriber's call origination feature, a vertical service feature, or a special service.This

25   feature is provisioned for each subscriber having basic POTS service.

2. Denied Origination Service—this request handler is provisioned for all subscribers that are prohibited from originating calls.

30

Call Authorization Feature

With reference to FIGURE 40, a call authorization feature determines whether or not a specific call origination attempt is authorized. These features typically invoke one

5   or more feature restriction components, which determine whether or not the specified call is subject to a restriction or subject to a class of restrictions, and report whether or not the call is authorized. Call authorization features may also provide alternative behaviors such as a PIN code authorization for specific call types.

10   Call authorization features are invoked as needed by origination request handlers and other features in order to determine whether or not a call is authorized. This may include circumstances wherein the call may occur at a later time. For example, a call authorization feature may be invoked to validate a call forwarding destinationnumber when the subscriber specifies the number.

15

Each subscriber is provisioned, by definition, with one call authorization feature. For example, FIGURE 40 shows the following three such feature:

1. No Call Restrictions—this feature simply approves all calls. This feature is

20   provisioned for all subscribers that have POTS Call Origination Service without any call restrictions.

2. Call Restrictions—this feature evaluates each provisioned restriction and reports the result. A call shall be authorized only if none of the provisioned restrictions

25   apply. This feature must be provisioned whenever a subscriber has one or more call restrictions.

3. Call Restrictions with PIN Authorization—this represents an advanced call authorization feature that includes a PIN based restriction override capability.

30

Call Restriction Feature

With reference to FIGURE 41, a call restriction feature, operating in conjunction
with subscriber attribute values, defines one or more prohibited calls or call types. A call
5   restriction feature is invoked as a subroutine to report whether or not a specified call is
prohibited.

Each subscriber may be provisioned with 0 or more call restriction features and
associated attribute values. For example, FIGURE X schematically illustrates the
10   following three call restriction features:

1. Code Restriction and Diversio

2. Toll Restricted
15

3. Outgoing Call Screenin

Call Origination Feature

20   A call origination feature originates and manages a call placed by a subscriber. A
call origination feature selects and invokes a call routing function, based on call type,
supports or directly provides call progress indications, and manages voice path.

Each subscriber may be provisioned with no more than one call origination
25   feature. For example, FIGURE 42 illustrates a POTS call origination feature,
provisioned for each subscriber having basic POTS service, that originates and approves
POTS calls.

Vertical Service Feature

A vertical service feature is invoked in response to a dialed vertical service code, e.g., "*70". Vertical service features may be part of a larger service, e.g., Cancel Call
5   Waiting, Activate Call Forward Busy Line, or may be a standalone feature, e.g., Customer Originated Trace (COTS).

A dialing plan defines zero or more associations between vertical services codes and vertical service features. Each subscriber should be provisioned with a dialing plan
10   that provides appropriate vertical service code to vertical service feature associations.

FIGURE 43 provides a diagram depicting the following exemplary vertical service features:

15   1. COTS—this feature reports information about the most recently received call.

2. Cancel Call Waiting—this feature temporarily de-activates the Call Waiting feature.
3. Call Pickup—this feature allows a subscriber to receive a call placed to
20   another directory number.

4. Calling Number Delivery Blocking—this feature temporarily marks the subscriber's directory number as private thus prohibiting its display to a called party.

25   **Termination Request Handler**

A termination request handler usually responds to a call termination attempt by choosing the most appropriate call termination feature from the subscriber's provisioned call termination features for the circumstances present at the time. A termination request
30   handler chooses a feature by evaluating each terminating feature's activation criteria in sequence according to feature precedence. The call termination feature whose activation

criteria are satisfied and has the highest precedence is invoked to terminate the call. However, a termination request handler may take direct action such as denying the call.

5 Each subscriber is provisioned with one termination request handler. By way of example, FIGURE 44 illustrates the following two termination request handlers:

1. POTS Termination Request Handler— this feature provides the standard "black telephone" response to a call termination request. This request handler selects and invokes the most appropriate call termination feature. This feature is provisioned for
10 each subscriber having basic call termination service.

2. Denied Termination Service—this termination request handler refuses all call termination attempts. This feature is provisioned for each subscriber that is prohibited from receiving calls.
15

Call Termination Feature

A call termination feature provides a specific call termination behavior. Call termination behaviors include presenting a call to the subscriber, forwarding a call,
20 refusing a call, etc.

FIGURE 45 schematically depicts the following exemplary call termination features:

25 1. Call Waiting—this feature presents a call to a subscriber when another call is already present and allows the subscriber to alternate conversation between the two calls.

2. Do Not Disturb—this feature, when active, refuses all calls.

3. Call Forward Busy Line—this feature, when active and the subscriber is busy at the time a call is presented, forwards the call to a previously specified directory number.

5   4. POTS Call Termination—this feature presents a call to the subscriber.

5. Call Forward All—this feature, when active, forwards all calls to a previously specified directory number.

10   6. Unidentified Caller Rejection—this feature, when active, refuses all calls wherein the calling party number is private, i.e., when the call's originator prohibits presentation of the calling party number.

Alerting Feature

15

An alerting feature notifies a subscriber that an inbound call is present. In one embodiment, there exists the following three sub-classifications of alerting features:

1. Idle Alerting Features—notify the subscriber by ringing the phone. These
20   features are invoked when the subscriber's handset is on hook.

2. Mid-Call Alerting Features—notify the subscriber by generating an audible tone during conversation. These features are invoked when the subscriber is connected with another call at the time the inbound call is presented.

25

3. Bounded Alerting Features—notify the subscriber that an inbound call is present and generate a "no answer" event if the subscriber does not "answer" within a pre-determined time limit. These features are further specialized into idle and mid-call altering features.

30

By way of example, FIGURE 46 depicts the following exemplary alerting features utilized in one embodiment of the invention:

5

1.   Basic Idle Alterting,

2.   Calling Number Delivery,

3.   Calling Name Delivery,

10   4.   Basic Call Alerting,

5.   Call Waiting with Calling Number, and

6.   Call Waiting with Calling Name.

15

Originating Feature Interactions

FIGURE 47 provides a diagram that depicts the relationship between boundary events, e.g., offhook, and the classifications of features. as well as relationships among
20   various originating feature classifications defined in one embodiment of the invention. In particular, FIGURE X shows how a feature handler resolves potential conflicts among different features. A feature handler, preferably implemented in a mark-up language, is chosen based on the feature passed to a context object by a subscriber terminal (or the network terminal).

25

For example, an Origination Request Handler is invoked in response to an off hook event. An Origination Request Handler employs a Dialing Plan for dialed number analysis and translation. An Origination Request Handler may invoke a Vertical Service Feature in response to a dialed vertical service code. Alternatively, an Origination
30   Request Handler may invoke a Special Service in response to a dialed special service

code, or may invoke a Call Authorization Feature in response to a dialed directory number.

A Call Authorization Feature may use one or more Call Restriction Features to

5   identify unauthorized calls or call types. An Origination Request Handler may invoke a Call Origination Feature to place an authorized call..

A Mid-Call Origination Request Handler may be invoked in response to an event, e.g., flash hook during a call. A Mid-Call Origination Request Handler utilizes a Dialing

10   Plan for dialed number analysis and translation. A Mid-Call Origination Request Handler may invoke a Vertical Service Feature in response to a dialed vertical service code. Alternatively, a Mid-Call Origination Request Handler may invoke a Special Service in response to a dialed special service code. Further, a Mid-Call Origination Request Handler may invoke a Call Authorization Feature in response to a dialed

15   directory number. A Mid-Call Origination Request Handler may invoke a Mid-Call Origination Feature to place an authorized call.

Terminating Feature Interactions

20   FIGURE 48 provides a diagram that depicts the inter-action of the terminating feature classifications. Similar to an origination request handler, a termination request handler processes call terminal events and ensures that no conflict arises among various features.

25   **Multi-Party Call Scenario**

The following example depicts the evolution of call and feature processing objects configuration starting with the origination of a two-party call and ending with a three-way call.

30

84

Subscriber Idle

FIGURE 49A provides a diagram that depicts the objects representing a particular subscriber in an idle state. With reference to both FIGURES 49A and 49B, a

5   SubscriberTerminal is a non-persistent object representing the current state of a subscriber. Each SubscriberTerminal contains a one-way reference to a persistent DirectoryNumber object. A DirectoryNumber object is the persistent representation of a particular subscriber including the subscriber's directory number string, the subscriber's provisioned features, and so forth. Each SubscriberTerminal also has a two-way

10   association with a ServicePort.

A ServicePort provides the associated SubscriberTerminal with an interface to the underlying switch. It converts the terminal's commands into MEGACO transactions and/or call signaling messages. It also converts MEGACO notifications and

15   call signaling messages into events for processing.

Subscriber Offhook

FIGURE 50 provides a diagram that depicts changes that occurr after the

20   subscriber lifts the handset to originate a call or invoke a vertical service feature element.

When an offhook event is observed and the subscriber is in the idle state, the SubscriberTerminal invokes the subscriber's origination request handler. An origination request handler is a provisioned feature that defines how to proceed when

25   offhook occurs from the idle state.

To execute the origination request handler, the SubscriberTerminal obtains the subscriber's origination request handler feature ID from the DirectoryNumber, and constructs an IndependentContext object, and passes the feature ID to this object.

30   Upon construction, the IndependentContext object obtains a pointer to the objects that

define the specified feature's state machine, initializes the processing engine, and executes the feature from the start state with the start event.

The IndependentContext has a two-way, *parent-child* relationship with the SubscriberTerminal. The parent/child relationship is established primarily for memory management and message routing purposes. Further, the call/terminal relationship reflects to whom each object sends messages (e.g., terminal to context object for new non-boundary events and/or context to terminal for commands). This association is established upon construction of the IndependentContext and supports the following capabilities:

1. The SubscriberTerminal is responsible for releasing the memory allocated to each IndependentContext it creates. However, a SubscriberTerminal cannot determine independently when an IndependentContext can be deleted.

   Each feature defines its own lifespan. An IndependentContext remains viable until each of its associated feature elements has concluded. Therefore, the last feature element to conclude must execute an action instructing the SubscriberTerminal to delete the IndependentContext. This instruction is always conveyed from *child to parent*.

2. Messages exchanged between originating and terminating features are routed from object to object using the SubscriberTerminal's IDT and CRV plus the destination feature's independent context ID. Each message is dispatched first to the SubscriberTerminal specified by the IDT and CRV values. The SubscriberTerminal examines the independent context ID and either processes the message, when the value is 0, or forwards the message to the appropriate IndependentContext, when the value is greater than 0.

The IndependentContext also has a two-way *call-terminal* relationship with the SubscriberTerminal. This association supports propagation of events and commands

related to call and feature processing. Events originating from the underlying CSX switch, received via the ServicePort, are propagated from *terminal to call*. Events originating from peer features are communicated directly to the destination independent context. Commands are always propagated from *call to terminal*.

5

Subscriber Dialed a Directory Number

FIGURE 51 provides a diagram that depicts the changes that occurred after the subscriber dialed a valid directory number.

10

Upon receiving the dial number, the "Orig Req Handler" calls upon the digit analyzer to classify the digit sequence. A subscriber may dial a destination code (directory number), a special service code, or a vertical service code depending upon the provisioned dialing plan. When the digit sequence is classified as a directory number and

15 the call is authorized, the "Orig Req Handler" invokes (calls) the subscriber's call origination feature. In this example, the subscriber's provisioned call origination feature is "POTS Call Orig".

As shown here, the "Orig Req Handler" has invoked the specified feature using a

20 "function call" model. As the name implies, the called feature is stacked on top the calling feature. Events are propagated from the top of the stack down. Each event is presented to each feature in this manner until a feature consumes the event, i.e., reacts to the event, or the end of the stack is reached. If the stack is exhausted, the event becomes stale. The SubscriberTerminal defines the handling of stale events.

25

Call Request Routed From Calling to Called Subscriber

FIGURE 52 illustrates a diagram depicting the changes that occurr after the called subscriber has been identified and presented with a call termination request.

30

87

The call origination feature calls the router to route the call to the appropriate destination; in this case another local subscriber. The router identifies and forwards a terminate call request to called subscriber. Upon receiving a terminate call request, a SubscriberTerminal invokes a termination request handler. A termination request
5   handler is a provisioned feature that defines how to proceed when a terminate call request is received.

Invocation of a termination request handler is similar to invocation of the "Orig Req Handler". In this scenario, the subscriber's termination request handler is POTS
10   Termination Request Handler. This particular request handler selects and invokes the appropriate call termination feature.

Note that a one-way *originator-terminator* relationship has been established from the called subscriber, the terminator, towards the calling subscriber, the originator. This
15   relationship supports message propagation between originator and terminator. However, it is one-way at this point because the originator has not yet learned the terminator's identity.

Note also that a *call-terminal* association has not yet been established between the
20   SubscriberTerminal and the IndependentContext. Many call termination features don't present the call to the subscriber; there are various features that refuse the call and various features that forward the call. Call termination features that don't present the call do not require control of the subscriber's access connection and therefore don't need to occupy that active call role. A feature only needs to occupy the active call role when and
25   if it needs control of the subscriber's access connection. Therefore, the termination request handler defers creation of the call-terminal association to the selected termination feature.

The CallDataBlock, depicted in this diagram as well as in previous diagrams,
30   describes a particular call leg between an originator and terminator. This object contains all attributes that must be shared between the originator and terminator; attributes such as

the originator and terminator ID information, call answer date and time, call disconnect date and time, and so forth. This object is carried with each message exchanged between originator and terminator to maintain synchronization and to provide up to date information.

5

Terminating Call Presented to Called Subscriber

FIGURE 53 presents a diagram that depicts an exemplary stable two-party call. A selected call termination feature, in this case POTS Call Termination, is invoked using

10   the function call model. This particular feature presents the call to the subscriber. It therefore establishes the *call-terminal* association, rings the phone, and sends a call proceeding message to the originator.

With the call proceeding message, the terminator's ID information passes to the

15   originator in the CallDataBlock and the originator-terminator relationship becomes bi-directional. This message also enables call progress indication to the originator; since this is a local subscriber-to-subscriber call audible ringing will be applied to the originating termination.

20   When the called subscriber answers, an answer indication passes from the terminator to the originator. This message carries the answer date and time and informs the originator that ringback can be terminated. At this point, a stable two-party call has been established.

25   **Calling Subscriber Executes a Mid-Call Origination**

FIGURE 54 presents a diagram that depicts the changes that occur after the originating subscriber issues a flashhook. An active call feature may consume a flashhook or the flashhook may represent a mid-call origination attempt. A mid-call

30   origination attempt is identified when, and if, a flashhook event becomes stale.

89

In this scenario, the flashhook event was propagated to the active call but was ignored because it is not meaningful to POTS Call Origination or to "Orig Req Handler". The event became stale so the SubscriberTerminal created an independent context and invoked the mid-call origination request handler; as seen here the subscriber's mid-call

5   origination request handler is "Mid-Call Orig Req".

By definition, an origination request handler takes control of the subscriber's audio connection in order to provide feedback (tones) to and collect information (digits) from the subscriber. A mid-call origination request handler is an origination request

10   handler. Therefore, "Mid-Call Orig Req" assumed the active call role.

In assuming the active call role with respect to the SubscriberTerminal, a mid-call origination request handler must also assume the role of terminal with respect to the existing call and inform the existing call that a multi-party feature is now present. The

15   behavior of a two-party feature is altered by the presence of a multi-party feature in that it must terminate unconditionally if and when the far end party hangs up. The mid-call origination request handler and any subsequent mid-call origination feature must know if and when the existing call terminates. By assuming the terminal role with respect to the existing call, the mid-call origination request handler places itself in the flow of

20   information pertaining to the existing call. Any subsequent mid-call origination feature inherits these associations; terminal with respect to the previously existing call and call with respect to the SubscriberTerminal, thus satisfying its need to be in the information flow regarding the existing call.

25   The respective features each retain responsibility for disposition of their respective call legs and call detail recording.

Mid-Call Origination Feature Invoked

FIGURE 55 illustrates a diagram that depicts the changes that occur after the subscriber dials a directory number.

5

The dialed digit sequence is classified as a destination code by the digit analyzer. Thus, the subscriber's mid-call origination feature is invoked. A mid-call origination feature is a provisioned feature that is invoked when a subscriber originates a call while participating in a two-party call. As shown here, the subscriber's mid-call originating

10 feature is three-way calling. A second context objectl, e.g., the context object of second call, serves as proxy terminal for the context object of the first call., thereby ensuring consistency in event processing.

The call origination aspects of a mid-call origination feature are similar to a two-

15 party call origination feature.

Call Routed to Called Subscriber

FIGURE 56 presents a diagram that depicts the changes that occur after the

20 second call is routed to the called subscriber.

Processing on the called subscriber's side is identical to that observed with the two-party call since the called subscriber is responding to a terminate call request. Features, object configuration, and associations are similar to those on the caller's side.

25

Terminating Call Presented to Called Subscriber

FIGURE 57 depicts the objects' configuration once the three-way call is completely established.

30 The original call's configuration remains unchanged except for the substitution of the SubscriberTerminal by the IndependentContext supporting the three-way call.

91

The three-way call feature acts as terminal to the previously existing call and as originator to the second call. The configuration of the two terminators is similar.

5 FIGURE 58 schematically illustrates selected processes and their interactions in a system of the invention. The inter-process communications can be provided by Corba orb, particularly within a TSP node. In addition, message transport between element management system (EMS) and a TSP node can also be provided by Corba orb.

10 The following appendices provide further details regarding the method and system of the invention.

Described above are systems and methods meeting the desired goals. Those skilled in the art will, of course, appreciate the illustrated embodiment is merely and example of the invention and that modifications can be made thereto without departing 15 from the scope of the invention. By way of non-limiting example, it will be appreciated that markup languages other than HTML, XML, CPML and so forth, as described above, may be utilized in practice of the invention. By way of further example, it will be appreciated that the methods and systems may be implemented with programming structures other than OOP objects.

20

Appendix A: Class 5 Feature List

This appendix provides an exemplary list of class 5 features that a TSP system of the invention can implement..

1. Basic 911 Service
2. CALEA/Wire Tap *(future release)*
3. 800 Number Service (800NS)
4. 900 Number Blocking
5. Abbreviated Dialing *(future release)*
6. Advance Do Not Disturb (ADND)
7. Automatic Callback (AC)
8. Automatic Recall (AR)
9. Call Forwarding Busy Line (CFBL)
10. Call Forwarding Don't Answer (CFDA)
11. Call Forwarding Variable (CFV)
12. Calling Number Delivery (CND)
13. Calling Number Delivery Blocking (CNDB)
14. Calling NAMe delivery (CNAM)
15. Call Transfer (CT)
16. Call Waiting (CW)
17. Cancel Call Waiting (CCW)
18. Denied Origination
19. Denied Termination
20. Distinctive Ringing/Call Waiting (DR/CW)
21. Do Not Disturb (DND)
22. Intercept treatment – announcement and tone
23. Local Number Portability (LNP)
24. Message Waiting Indication (MWI) – not supported in the initial release, no plan for voice mail feature.
25. Multi-line hunt group
26. Multi-PIC
27. Operator Service

28.   Selective Call Forwarding (SCF)

29.   Selective Call Reject (SCR)

30.   Series Completion

31.   Speed Calling (SC)

5   32.   Three-Way Call

33.   Toll Restriction

34.   Un-identified Call Reject (UCR)

Appendix B: Feature Analysis

This appendix provides a brief description of the a number of services listed in appendix A. The descriptions are extracted from Bellcore LSSGR.

Feature Name	Abbreviation	FSD	Access Code	Availability
800 Service	**INWATS**	**02-01-0020**	**n/a**	**all**

5   *Extracted from LSSGR – begin*

This feature is a terminating-only service that allows a subscriber to receive message calls originating within specified servic e areas, with the call charges billed to the called party instead of the calling party. The service is provided to one or more dedicated access lines.

The 800 service billing information and appropriate service measurements are registered by the Automatic

10   Message Accounting (AMA) recording arrangement serving the terminating exchange. When the billing arrangement used by the switching system is via a ``connection to Centralized Automatic Message Accounting (CAMA) offices'' (55-01-0300), the 800 service billing information is recorded on auxiliary equipment separate from the CAMA billing arrangement.

Extracted from LSSGR – end

15   *Event:*   800NS (DialComplete + dialed number is prefixed with 800)

Seizure

Action: 800NS

Initiate an 800 service TCAP query and starts a timer

20   When receiving a successful TCAP response, return the billing DN and routing DN

and terminate the feature

When receiving a failed TCAP response, provide the instruction for intercept

treatment and terminate the feature

If the TCAP query timer expires before a response is received, provide the instruction

25   for intercept treatment and terminate the feature

Seizure

Generate AMA record if the access line is of type INWATS

Interaction: **INWATS-AR**

30   Activation of AR is not allowed on INWATS line.

95

Feature Name	Abbreviation	FSD	Access Code	Availability
911	911	15-01-0000	n/a	all

Extracted from LSSGR– begin

The basic 911 emergency service feature provides a 3-digit universal telephone number (911) that gives the

5   public direct access to an emergency service bureau, usually without charge to the calling subscriber. The
emergency service is one-way only, terminating to the service bureau. The emergency service bureau is
usually located within a police department, although in some communities it is located in a fire department
or in an independent agency serving as a communications center. A given local switching system should
serve no more than one emergency service bureau.

10   When the originating line and the emergency service bureau are served by the same switching system, the
bureau has the capability of holding and disconnecting the connection and monitoring the supervisory state,
and ringing the originating station back.

When the local switching system is in an area with expanded emergency service (E911) served through a
tandem switch, the emergency call is advanced to the tandem switch with calling line ANI.

15   After the attendant answers a line originated call and the conversation path is established, the calling line
should not be released even if the disconnect signal is received from the calling, until a disconnect signal is
received from the service bureau. It is also desirable to release the call after approximately 45 minutes
have elapsed since the caller disconnect signal has been received. This time-out period begins again on
receipt of an emergency ring back request from the operator (optional service). On time-out, the calling

20   line should be idled, reorder tone should be applied to the service bureau and a message printed to inform
telephone company personnel of the condition.

A disconnect signal from the service bureau should result in a forced-disconnect of an established
connection at any time, independent of the calling line status. Forced-disconnect should occur immediately
on detection of a valid disconnect signal from the service bureau for both line and trunk originated calls.

25   On trunk originated 911 calls, an on-hook signal should be sent toward the calling end. Normal called
party timed disconnect in the originating office should then release the trunk portion of the connection.
Extracted from LSSGR - end

Event: 911 (DialComplete + dialed string =='911')

30

Action: Enable the localRelease event
                  Enable the remoteRelease event
                  When the localRelease event is detected, apply ring signal on the originator local loop
                  and start a 45-minute timer.

35   Upon expiration of the 45-minute timer, release the call.

When the remoteRelease event from the 911 operator is received, cancel the timer, disable the localRelease and remoteRelease events, and terminate the 911 feature.

Interaction: **911 – 4_ and 8_ party line**

On ringback, all parties should be rung, interleaved, until someone answers.

911-DOS (Denied Originating Service)

If these two features are allowed to be assigned to the same line, DOS should take precedence.

911-DTS (Denied Terminating Service)

Ringback should be performed when requested by the service bureau.

911- Code Restriction and Diversion

911 should not be screened out, so a line with code restriction can always call 911.

911- Series Completion

The switching system should not be able to series complete to 911.

911- CFBL

CFBL forwarded DN cannot be 911.

911- CFDA

CFDA forwarded DN cannot be 911.

911- CFV

CFV forwarded DN cannot be 911.

911- Make Busy Key

Ringback should be performed regardless of the make busy condition.

911- CW

During a call to 911 attendants, CW is inhibited.

911- TWC (including usage sensitive TWC)

During a call to 911 attendants, TWC cannot be used.

In a normal 2-way call, a TWC subscriber places the call on consultation hold and initiates a 911 call. It is desirable that the TWC subscriber can flash once to establish a TWC connection. All subsequent flashes should be ignored.

5   Design note: 911 FLO should handle all call progress events so that it can control the handling of these events.

Feature Name	Abbreviation	FSD	Access Code	Availability
Selective Call Rejection	**SCR**	**01-02-0760**	***60 or *80**	**Subscription Usage**

Extracted from LSSGR – begin

Selective Call Rejection allows customers to block subsequent calls from a calling DN after receiving a

5   harassing call. After receiving an undesired call, the customer dials a code that places the last calling

party's DN on a list for rejection. Subsequent calls from the rejected party are intercepted and routed to an

announcement. Selective call rejection also enables customers to enter additional DNs onto the list in a

manner similar to selective call forwarding or VIP alert (01-02-1270). Calls from these parties are also

intercepted and routed to a rejection announcement.

10   *Extracted from LSSGR – end*

     Event: Seizure

     Action: Compare the calling number with the SCR list

                Play the SCR announcement/tone and release the call if the calling DN is on the list

15   Continue the call termination treatment otherwise

                Note: the administration of the SCR list is part of SLE, and therefore it uses the

                procedures defined for SLE.

     Interaction: SCR takes precedence over other features at seizure event. If a call is rejected due to the

20   calling number is on the SCR list, all other features at seizure event are disabled.

     SCR-AC

     SCR customers can activate Automatic Callback (AC) to numbers on their SCR lists.

     Ringback for such activations is never blocked; rather, it is given to the customer.

25

     SCR-AR

     An incoming call with calling number on SCR list will be rejected and the incoming

     memory slot will not be updated.

30   **SCR-SCF**

     SCR overrides SCF.

Feature Name	Abbreviation	FSD	Access Code	Availability
1.1.1.1 Do Not Disturb	**DND**	**01-02-0750**	**n/a**	

Extracted from LSSGR – begin

The Do Not Disturb feature is activated/deactivated by the customer. It routes calls destined to the customer's DN either to a special do not disturb announcement or to a special tone. The announcement may return answer supervision. An option of the service provides that the calling party may dial a prearranged Personal Identification Number (PIN) while listening to the announcement/tone. The local switching system collects the DTMF digits and, if they correspond to the customer-specified number, the calling party will override the do not disturb function and receive normal terminating treatment. The customer may dial the activation code to enable this service, dial the deactivation code to disable the service, or dial the PIN change code to initially enter or modify the number.

Extracted from LSSGR - end

Event: Seizure

Action: DND Meta-feature

Basic treatment –

Play the DND announcement and release the call

Advanced treatment -

Play the DND + collect PIN announcement/tone to prompt of user input

Collect digits

Allow the call to go through if the collected digits match the PIN

Reject the call otherwise

Interaction:

Feature Name	Abbreviation	FSD	Access Code	Availability
Cancel Call Waiting	**CCW**	**01-02-1204**	***70**	**Subscription Usage**

Extracted from LSSGR - begin

Cancel call waiting is a feature that allows the customer with CW service to inhibit the operation of CW for one call. The customer dials the cancel call waiting code, obtains recall dial tone, and places a call

normally. During this call, CW service should be inactive so that anyone calling the CW customer should receive the normal busy treatment, and no CW tones should interrupt the customer's call.

Extracted from LSSGR - end

5   *Event:*   DialComplete

     Action: CCW Meta-feature

                       At DialComplete detect CCW

                       Remove CW FLO from busy event stack

10   Enable local release event

                       Upon detection of local release, place CW FLO back to busy event stack

                       Terminate

     Interaction: ***CCW-TWC***

15   If the initiator of TWC hangs up with a party on hold, he/she should be rung back and connected to the held party on answer. If the initiator's CW was disabled prior to hanging up on the held party, it should remain disabled after the customer answers the ringback.

Feature Name	Abbreviation	FSD	Access Code	Availability
Call Waiting	**CW**	**01-02-1201**		**Subscription Usage**

Extracted from LSSGR – begin

CW is a feature whereby a line in the talking state is alerted by a call waiting tone when another call is
attempting to complete to that line. The call waiting tone is only audible to the line with the CW feature
activated. Audible ringing is returned to the originating line. The service also provides a hold feature that
is activated by a switchhook flash. Consecutive flashes allow the customer (with the service activated) to
alternately talk to the original and the new calling party. If the customer with the service activated hangs
up while one party is on hold, the customer with the service activated is automatically rung back, and upon
answer is connected to the held party.

Extracted from LSSGR – end

Event: Busy

Action: Apply the call waiting tone

                 Enable hookflash, localRelease, and remoteRelease events

                 At hookflash event, swap the speech path to the other call leg

                 At localRelease event, disconnect the remote talking party, apply ringing signal to the

                 endpoint, and enable answer event.. At the subsequent answer event, disable hookflash,

                 localRelease, remoteRelease, and answer events, and terminate the feature.

                 At remoteRelease event, swap the speech path to the other call leg if the release event is

                 from talking party. And then disable hookflash, localRelease, and remoteRelease events.

                 Finally terminate the feature.

Interaction: **CW- SC (Series Completion)**

                 A new call terminated on a CW user, if there is only one existing call

                 terminated on the user's line and the call is in talk stage, CW overrides

                 SC.

                 A new call terminated on a CW user, if there is only one existing call

                 terminated on the user's line and the call is not in talk stage, SC overrides

                 CW.

                 A new call terminated on a CW user, if the user uses *Make-Busy* key to

                 become busy, SC overrides CW.

A new call terminated on a CW user, if there are more than one existing calls terminated on the user's line, SC overrides CW.

CW- CFBL

If a line has CW and CFBL, CW should take precedence, unless
1. the existing call is not in talk state
2. the *Make-Busy* key is used by the user to become busy
3. there are more than one calls on the line

CW- CFDA

CW and CFDA can be assigned to a line and be active simultaneously. When a line is busy and in talk state, an additional call is offered to the line, the call should be offered under CW and CFDA timer should be started. If the CFDA timer expires before the user answers, the additional call should be forwarded.

CW- *Make-Busy* Key

The Make-Busy key should take precedence over CW. If CFBL or Series Completion is also assigned to the line, see the interaction defined for these two features.

CW- CCW

CW and CCW should not be active simultaneously.

CW-TWC

If both features are assigned to a line, the flash after the call waiting tone should be treated as the request to initiate the TWC. Also if the party to whom the customer was talking hangs up and another is held, the flash should be recognized as a request to establish a talking connection with the held party.

The initiator of TWC should not receive any CW treatment while in a TWC mode or while a party is on hold.

When a user is in a TWC, but not the TWC initiator, CW overrides TWC. The additional call attempt should be treated using the CW treatment.

CW-USTWC

5

The initiator of TWC with USTWC activated should not receive Call Waiting(CW) calls.
The other parties to TWC should be able to receive CW Calls.
In flash interpretation, CW should take precedence over USTWC.

CW-911

10

During a call to a 911 attendant, the CW feature should not take place.

CW-Operator service

CW should take precedence over operator recall when the system detects a flash after CW tone is applied to the connection. Subsequent flashes control CW; no operator

15   recall should be allowed until a two-way connection is reestablished.

Appendix C

This appendix provides an exemplary supplementary document type definition for
CPML. It further provides examples of CPML+ scripts for call control, and an example
5 of an endpoint's feature subscription in CPML+.

Supplementary Document Type Definition for CPML

```
     <!ENTITY %  result 'result          (true|false)          "true"'>


10   <!ENTITY %  eventId              'id      (OFFHOOK | ONHOOK |
     DIGITS|PROCEEDING|
     ALERT|ANSWER|LOCALRELEASE|
     REMOTERELEASE|
     ......)            #REQUIRED '>
15

     <!ENTITY %  param            'param          (CDATA|none)
          "none"'>
     <!ENTITY %  testMethod       'method          (testDNISPrefix|
          testDNISPostfix|
20                                           testDNISSubString|
     testCFBLActive|
                                          testCFDAActive|          ....)
                     #REQUIRED'>
     <!ENTITY %  execMethod       'method          (makeCall|
25   collectDigit|

                                          acceptCall|
     answerCall|

                                          releaseCall
     abortCall|

30                                         applyTone|
     collectDigits|

                                          playAnnouncement|          ....)
                     #REQUIRED'>


35   <!ELEMENT    script          (state+)>
     <! ATTLIST   script          id          (CDATA)     #REQUIRED
     billingId          (CDATA)     "none">
```

```
        <!ELEMENT    state             (event+) >
        <! ATTLIST   state             id            (CDATA)        #REQUIRED>


   5    <!ELEMENT    event             (do*)>
        <!ATTLIST    event             %eventId;
        billingId         (CDATA)      "none">


        <!ELEMENT    do                (test?, (execute|link)+, nextState?)>
  10    <!ELEMENT    test>
        <!ATTLIST    test              %testMethod;
                                       %param;
                         %result;>


  15    <!ELEMENT    execute>
        <!ATTLIST    execute                   %execMethod;
                                       %param;
        billingId         (CDATA)      "none">


  20    <!ELEMENT    nextState>
        <! ATTLIST   nextState         id            (CDATA)        #REQUIRED>


        <!ELEMENT    loadScript>
        <! ATTLIST   loadScript        target             (CDATA)        #REQUIRED
  25    data         event|none) "none">


        <!ELEMENT    userProfile       (event+)>
        <! ATTLIST   userProfile       name          (CDATA)        #REQUIRED
        uid          (CDATA|none)      #REQUIRED
  30    group        (CDATA|none)      "none"
        system            (CDATA)      #REQUIRED>
```

Examples of Call Control CPML+ scripts

```
<script id="CallControlScript">

5   < -- This script has one single state />

    <state id="idle">

      <event id="offhook">
10      <do>
        <loadScript target=" OriginationScript" data=event/>
        </do>
      </event>

15    <event id="seizure">
        <do>
        <loadScript target="TerminationScript" data=event/>
        </do>
      </event>
20
    </state>

</script>

25  <script id="OriginationScript">

    < -- Implements Origination Basic Call Model />

    <state id="idle">
30
      <event id="offhook">
        <do>
        <execute method="timeStampOffhook" param="none"/>
        <execute method="processOffhookEvent" param="none"/>
35      </do>
        <do>
        <test method="testDNISReceived" param="none" result="true"/>
        <execute method="routeCall">
```

```
           <nextState id="awaitAnswer "/>
           </do>

         </event>
5
       </state>

       <state id=" awaitAnswer">

10       <event id="remoteAnswer">
           <do>
           <execute method="timeStampRemoteAnswer" param="none"/>
           <execute method="connectSpeechPath" param="none"/>
           <nextState id="connect "/>
15         </do>
         </event>

          <event id="release">
           <do>
20         <execute method="timeStampRelease" param="none"/>
           <nextState id="_terminate"/>
           </do>
         </event>

25     </state>

        <state id="connect">

         <event id="release">
30         <do>
           <execute method="timeStampRelease" param="none"/>
           <execute method="releaseRemoteParty" param="none"/>
           <execute method="disconnectSpeechPath" param="none"/>
      <nextState id="_terminate"/>
35         </do>
         </event>

         <event id="remoteRelease">
```

```
        <do>
        <execute method="timeStampRelease" param="none"/>
        <execute method="localRelease" param="none"/>
        <execute method="disconnectSpeechPath" param="none"/>
5   <nextState id="_terminate"/>
        </do>
      </event>


    </state>
10

</script>
```

An example of an endpoint's feature subscription in CPML+

```
    < -- CPML for subscriber Simon St.  Laurent - with  Message Waiting,
15  Call Forwarding Don't Answer and Call Waiting  features />

    <useProfile name="Simon St.  Laurent"
    uid="abcd-1234-gbdc-4566-143333"
    group="St.  Laurent Family"
20  system="LATA101-Area2">
        <event id=offhook>
          <do>
            <execute method="incrementOffhookCount">
          </do>
25        <do>
            <test method="isMessageWaiting" result="true"/>   ←We can use
    else structure here
            <execute method="applyTone" param="stutterTone">
            <nextState id="_terminate"/>
30        </do>
          <do>
            <test method="isMessageWaiting" result="false"/>
            <execute method="applyTone" param="dialTone">
            <nextState id="_terminate"/>
35        </do>
      </event>

      <event id=911Event>
```

109

```
            <do>
             <execute method="runFSM" param="911FSM">
             <nextState id="_terminate"/>
            </do>
5          </event>


           <event id=dialComplete>
            <do>
                <test method="is800Dialed" result="true"/>
10              <execute method="runFSM" param="800FSM">
                <nextState id="_terminate"/>
            </do>
            <do>
                <test method="is900Dialed" result="true"/>
15              <execute method="releaseCall" param="900 number blocked">
                <nextState id="_terminate"/>
            </do>
            <do>
                <test method="isCFDAActivationDialed" result="true"/>
20              <execute method="runFSM" param="CFDAActivationFSM">
                <nextState id="_terminate"/>
            </do>
            <do>
                <test method="isCFDADeactivationDialed" result="true"/>
25              <execute method="runFSM" param="CFDADeactivationFSM">
                <nextState id="_terminate"/>
            </do>
           </event>


30         <event id=CFDA_Timer>
             <do>
                <execute method="routeCall" param="5551212">
                <nextState id="_terminate"/>
             </do>
35         </event>


           <event id=busy>
             <do>
```

```
            <execute method="runFSM" param="CWFSM">
            <nextState id="_terminate"/>
        </do>
    </event>
</useProfile>
```

Appendix D

This appendix presents call waiting CPLM+ scripts, and a call waiting logic that can be better understood with reference to FIGURE 34.

Example of Call Waiting CPML+ scripts

5

```
<script id="CallWaitingFeature">

 <state id="Initial">

10      <event id="busy">
         <do>
         <execute method="incrementCallWaitingAttempt" param="none"/>
         <execute method="applyTone" param="CW"/>
         <execute method="enableCallEvent" param="hookflash, localRelease,
15 remoteRelease"/>
                       <execute method="setFirstCallLegHandle"
param="none"/>
         <execute method="setFirstCallLegActive" param="none"/>
<nextState id="Alert"/>
20       </do>
        </event>


    </state>

25   < state id="Alert">

       <event id="hookflash">
         <do>
         <execute method="setSecondCallLegHandle" param="none"/>
30  <execute method="swapToSecondCallLeg" param="none"/>
             <execute method="setSecondCallLegActive" param="none"/>
             <nextState id="Conn"/>
         </do>
        </event>
35
       <event id="localRelease">
         <do>
```

```
                <execute method="disableCallEvent" param="hookflash,
localRelease, remoteRelease"/>
                    <nextState id="_terminate"/>
            </do>
5       </event>


        <event id="remoteRelease">
          <do>
            <execute method="disableCallEvent" param="hookflash,
10   localRelease, remoteRelease"/>
                    <nextState id="_terminate"/>
            </do>
        </event>


15   </state>


    < state id="Conn">

        <event id="hookflash">
20         <do>
            <test method="isFirstCallLegActive" result="true"/>
<execute method="swapToSecondCallLeg" param="none"/>
                <execute method="setSecondCallLegActive" param="none"/>
            </do>
25         <do>
            <test method="isSecondCallLegActive" result="true"/>
<execute method="swapToFirstCallLeg" param="none"/>
                <execute method="setFirstCallLegActive" param="none"/>
            </do>
30       </event>


        <event id="localRelease">
          <do>
            <execute method="releaseRemoteTalkingParty" param="none"/>
35         <execute method="enableCallEvent" param="offhook"/>
                <nextState id="Rung"/>
            </do>
        </event>
```

113

```
        <event id="remoteRelease">
          <do>
          <test method="isReleasingPartyTalking" result="true"/>
    <execute method="connectToHeldParty" param="none"/>
          </do>
          <do>
          <execute method="disableCallEvent" param="hookflash,
    localRelease, remoteRelease"/>
                  <nextState id="_terminate"/>
          </do>
        </event>


      </state>

      <state id="Rung">

        <event id="offhook">
          <do>
          <execute method="disableCallEvent" param="hookflash,
    localRelease, remoteRelease,
          offhook"/>
    <nextState id="_terminate"/>
          </do>
        </event>

      </state>

    </script>
```

Call Waiting Feature Logic

```
(CallWaitingLogic
        //Initial state
5       (Initial   (       busy
(       ()      //predicate
(       //method
                        incrementCallWaitingAttempt();
        setFirstCallLegHandle()
10      setFirstCallLegActive()
        applyTone(cwTone)
                        enableCallEvent(hookflash, localRelease,
remoteRelease)
)
15                      (Alert)     //next state
                )
                )
        ) //end of Initial state

20      //Alert state
        (Alert    (       hookflash
(       ()      //predicate
(       //method
        setSecondCallLegHandle()
25      swapToSecondCallLeg()
        setSecondCallLegActive()
)
                        (Conn)      //next state
                )
30              )
                (       localRelease
(       ()      //predicate
(       //method
                        disableCallEvent(hookflash, localRelease,
35  remoteRelease)
)
                        (_terminate)        //next state
                )
```

```
                               )
                            (        remoteRelease
        (       ()      //predicate
        (       //method
5                               disableCallEvent(hookflash, localRelease,
        remoteRelease)
        )
                               (_terminate)        //next state
                            )
10                          )
             ) //end of Alert state


             //Conn state
             (Conn       (       hookflash
15       (       (isFirstCallLegActive())       //predicate
         (       //method
             swapToSecondCallLeg()
             setSecondCallLegActive()
         )
20                             (Conn)        //next state
                            )
         (       (isSecondCallLegActive())        //predicate
         (       //method
             swapToFirstCallLeg()
25       setFirstCallLegActive()
         )
                               (Conn)        //next state
                            )
                            )
30                          (        localRelease
         (       ()      //predicate
         (       //method
                               releaseRemoteTalkingParty()
                               enableCallEvent(offhook)
35       )
                               (Rung)        //next state
                            )
                            )
```

116

```
                                (        remoteRelease
         (        (isReleasingPartyInTalk())     //predicate
         (        //method
                                connectToHeldParty()
    5    )
                                ()      //next state
                         )
         (        ()                        //predicate
         (        //method
   10                            disableCallEvent(hookflash, localRelease,
         remoteRelease)
         )
                                (_terminate)       //next state
                         )
   15                         )
                ) //end of Conn state


                //Rung state
                (Rung        (        offhook
   20    (        ()       //predicate
         (        //method
                                disableCallEvent(hookflash, localRelease,
         remoteRelease,
           offhook)
   25    )
                                (_terminate)        //next state
                         )
                         )
                ) //end of Rung state
   30
         )     //end of CallWaitingLogic
```

Appendix E: TMN Compliance

Telephone service providers, e.g., CLEC and ILEC, use many Operations Systems (OSs) to provide centralized support for the maintenance and administration of

5   the telephony network equipment. More often than not, the network equipment are obtained from different vendors all over the world. Thus, the tasks of incorporating heterogeneous network equipment and providing seamless end-to-end services to customers have become very challenging. Telecommunications Managed Network (TMN) recommendations are defined by ITU in the M.3000 series to provide a

10   framework for a standardized mechanism to manage the networks. TMN enables the service providers to achieve interoperability across all managed networks over heterogeneous network equipment.

In the TMN architecture, a Network Element (NE) contains manageable information of a network equipment that is monitored and controlled by an OS. In order

15   to be managed within the scope of TMN, an NE must have a standard TMN interface, which is the Common Management Information Protocol (CMIP). If an NE does not have a standard interface, the NE can still be managed via a Q-Adapter. Currently two Q-adapters are supported; TL1 Q-adapter and SNMP Q-adapter.

SNMPv3 has been chosen as the network management protocol for TSP and

20   CSX, which renders the TSP/CSX system a TMN manageable network equipment. SNMPv3 offers the acknowledgement mechanism to the message originator, which is required in the TMN architecture.

With reference to FIGURE 35, TMN comprises layers of building blocks that provide overall epitome of the management issues and functions of TMN. TMN includes

25   a network element layer, an element management layer, a network management layer, a service management layer, and a business management layer.

The network element layer presents the manageable information that is managed and controlled by OSs. The MIBs of TSP/CSX, the SNMP proxy agent on TSP, and the SNMP Q-adapter reside in this layer. Through the SNMP Q-adapter, TSP and CSX are

30   TMN enabled. TSP/CSX will be SNMPv3 compliant.

The element management layer manages one or multiple NEs. This layer deals with, for example, the configuration data, logs, statistics of one or multiple network elements. The Network Management System (NMS) in TSP provides the functions of this layer. In some embodiments, however, the element management layer will not be TMN conformant. That is, it does not provide Q3 interface to communicate with the network element layer and network management layer. NMS is the tool to provision the operation and support of TSP/CSX.

The network management layer has the managed view of the entire network. It coordinates all network activities and supports the service management layer functions via Q3 interface.

The service management layer utilizes information presented by network management layer to manage contracted service to existing and potential customers. This is the basic point of contact with customers for provisioning, accounts, quality of service, and fault management. It is also the key point for interactions with service providers and other administration domains. This layer communicates with the network element layer and element management layer via Q3 interface.

The business management layer includes high-level planning, budgeting, goal setting, executive decisions, business level agreements, etc.

Appendix F

       This Appendix provides an example of a reference lexical structure for defining a state machine.

5

```
<!--
========================================================================
=========================
Element     : LOGIC
Summary     : A logic represents either a state machine or a rules
machine structure.

            This element exist to let us handle both state and rules
machines in the same fasion.

            A rules set specification defines the behavior of a stateless
logic.

            A finite state machine specification defines the behavior of
state-full logic
            elements.
-->
<!ELEMENT LOGIC ( FSM ) >
<!ATTLIST LOGIC Name NMTOKEN #REQUIRED >



<!--
========================================================================
=========================
  Element     : FSM (Finite State Machine)

  Summary     : The State Machine is composed of a collection of states.
These states specify behavior
            that defines the state machine.  The state machine CPML
element has an attribute
            of the initial state for this machine.

  Validation : The inital state attribute shall be validated against the
child elements of
            FSM to make sure that the initial state is defined.
-->
<!ELEMENT FSM ( (STATE+, END_STATE*) ) >
<!ATTLIST FSM InitState NMTOKEN #REQUIRED >



<!--
========================================================================
=========================
  Element     : STATE

  Summary     : A State is composed of a collection of transitions.  Each
state element has a
            name attribute which is used to identify the state.
```

10

15

20

25

30

35

40

45

50

120

```
        Example     : This is an example of a state with the logical name
        Initial.

                    <STATE Name="Initial">
 5      -->
        <!ELEMENT STATE ( TRANSITION+ ) >
        <!ATTLIST STATE Name NMTOKEN #REQUIRED >


10      <!--
        ============================================================================
        ========================
           Element     : END_STATE

15         Summary     : An end state is a state that has no transitions.  Each
        state element has a
                        name attribute which is used to identify the state.

           Example     : This is an example of a end state.
20
                        <END_STATE Name="Stop">
        -->
        <!ELEMENT END_STATE EMPTY >
        <!ATTLIST END_STATE Name NMTOKEN #REQUIRED >
25
        <!--
        ============================================================================
        ========================
           Element: TRANSITION
30
           Summary: A transition belongs to a state.  These are the pathways out
        of that state, given
                        an event.  If we get an event we will take the transition
        associated with that event
35                      from the current state to another state.  A transition has a
        set of sequences. Once
                        a sequence is able to execute successfully we will stop testing
        the other sequences,
                        and will transition to the next state defined by that
40      sequence.

           Example: If we are in the "S1" state and we get an event "OneBit" that
        event will trigger the
                        execution of the transition associated with that event. Once
45      the transition is executed
                        the behavior ("Actions") associated with that transition will
        be executed.

                        <STATE Name="S1">
50                          <TRANSITION Name="T1_1" Event="OneBit">
                                <FSMSEQUENCE NextState="Odd">

           Validation: The Event name shall be validated against a list of known
        events.
55      -->
        <!ELEMENT TRANSITION ( FSMSEQUENCE+ ) >
        <!ATTLIST TRANSITION Name NMTOKEN #REQUIRED >
```

```
<!ATTLIST TRANSITION Event NMTOKEN #REQUIRED >
```

5
```
<!--
===========================================================================
========================
   Element: FSMSEQUENCE
```

10
```
   Summary: A sequence is the set of all possible actions for the current
event.
            An optional predicate or guard condition is used to distinguish
between
            the different sequences.
```
15

```
            The predicate refers to a method that yields a boolean "true"
or "false" result.
            The predicate sequence shall be executed until the first one
fails. In other words
```
20
```
            the results of the predicates are anded together.

            If all of the predicates returned true we will execute the
sequence. Otherwise
            we will continue going through the sequences until we find one
```
25
```
that is successful.

            A sequence with no predicate guards is always executed, and any
remaining sequences
            are never invoked.
```
30
```
            The predicates function like other actions they will invoke a
method on the current
            focal point.
```
35
```
            The sequence has an attribute that corresponds to the name of
the next state.

   Validation: Predicate methods shall be validated against the core
object model specification.
```
40
```
            The sequences next state attribute will be validated
against the specified
            state names.
```

```
   Example:  In this example the sequence does not have any predicates.
```
45
```
So it will execute
            the current sequence.  If all of the actions are successful we
will transition to
            the state named "Odd".
```

50
```
                <FSMSEQUENCE NextState="Odd">
                <ACTIONS/>
```

```
   Example 2: This is a place holder for an example that makes use of
predicates.  Currently predicates are
```
55
```
            not implemented in our prototype, so they have not been
included yet...
-->
```

```
<!ELEMENT FSMSEQUENCE ( PREDICATES?, ACTIONS+ ) >
<!ATTLIST FSMSEQUENCE NextState NMTOKEN #REQUIRED >
```

5 `<!--`
```
====================================================================
========================
     Element: ACTIONS
     Summary: An Action is a container for all of the actions that are
10 associated with this sequence.
                 The set of actions is what gives a sequence its behavior.

                 It can contain any type of action.

15                 A transform or a call action has to be the last element in
the actions sequence.

     Example: In the below example you can see how the actions are
structured.
20                 <FSMSEQUENCE NEXTSTATE="Odd">
                     <ACTIONS>
                         <STR_ASSOCIATION FocalPoint="ParityChecker"/>
                         <ACTION NAME="print1BitAction"/>
                         <ACTION NAME="printOddAction"/>
25                     </ACTIONS>
                 </FSMSEQUENCE>

ACTIONS ( ACCESSOR | ACTION | ASSOCIATION | STR_ASSOCIATION |
TRANFORMATION | CALL_ACTION | LITERAL)*
30
     -->
     <!ELEMENT ACTIONS ( (ACCESSOR | ACTION | ASSOCIATION | STR_ASSOCIATION |
LITERAL)* |
                         ((ACCESSOR | ACTION | ASSOCIATION | STR_ASSOCIATION |
35 LITERAL)*,
                         (TRANSFORMATION | CALL_ACTION) )) >
```

40 `<!--`

```
====================================================================
========================

45    Element: PREDICATES

     Summary: An Action is a container for all of the actions that are
associated with this sequence.

50     Example: In the below example you can see how the actions are
structured.



55 PREDICATES ( ACCESSOR | ACTION | ASSOCIATION | STR_ASSOCIATION |
TRANFORMATION | CALL_ACTION | LITERAL)*
```

```
-->
```

5   `<!ELEMENT PREDICATES ((ACCESSOR | ACTION | ASSOCIATION |
    STR_ASSOCIATION | LITERAL)*) >`

10
```
<!--
========================================================================
========================
   Element: ACTION
```
15
```
   Summary: An Action represents a small atomic piece of behavior.  The
name attribute on action
                refers to a method on the current focal point.  The focal
point is the object that
```
20
```
                the actions are called on.  When the action is executed it
will call the
                corresponding method on its focal point.
```

```
   Validation: Action names shall be validated against the core object
```
25
```
model specification, to
                verify that the method exists.
```

```
   Example:
                In the below example,  we will call the apply dial tone method
```
30
```
on the current
                focal point of the engine.  In this case the current focal
point will have a
                method that applies the dial tone.
```

35
```
                <ACTION Name="applyDialTone"/>
ELEMENT ACTION EMPTY
-->
<!ELEMENT ACTION ( ACCESSOR | ACTION | ASSOCIATION | STR_ASSOCIATION |
LITERAL)* >
```
40 `<!ATTLIST ACTION Name NMTOKEN #REQUIRED >`


```
<!--
========================================================================
```
45
```
========================
   Element: ACCESSOR
```

```
   Summary: An ACCESSOR I think is a remanent now.  All actions are able
to pass parameters
```
50
```
                back and forth.  Currently it is just treated like other
actions.
```

```
   Example:
                In the below example,  we will call the apply dial tone method
```
55
```
on the current
                focal point of the engine.  In this case the current focal
point will have a
```

```
                    method that applies the dial tone.

                        <ACTIONS>
                            <ACCESSOR Name="getDialTone"/>
 5
                Validation: Accessor names shall be validated against the core object
            model specification, to
                            verify that the method exists.
                --->
10          <!ELEMENT ACCESSOR EMPTY >
            <!ATTLIST ACCESSOR Name NMTOKEN #REQUIRED >


            <!--
15          ======================================================================
            =========================
                Element: ASSOCIATION

                Summary: An association will move the focal point ( see Action ), to a
20          new location.
                        The name of the element will correspond to a method on the
            current focal point.
                        It will move the focal point to the result of that method.


25          Validation: Association names shall be validated against the core
            object model specification,
                        to verify that the method exists.


                Example:
30                      In the below example,  we will call the getTerminal method on
            the current focal point.
                        We will then set the focal point of engine, to the returned
            value.  In this case the
                        current focal point will have a method that gets the terminal.
35
                        <ACTIONS>
                            <ASSOCIATION Name="getTerminal"/>
                --->
            <!ELEMENT ASSOCIATION EMPTY >
40          <!ATTLIST ASSOCIATION Name NMTOKEN #REQUIRED >


            <!--
            ======================================================================
45          =========================
                Element: STR_ASSOCIATION

                Summary: This kind of string association is used for testing.  It will
            eventually be removed.
50                      I can pass in the name of the new focal point ( see Action ),
            that will be looked
                        up instead of having to call a method to get the new focal
            point and then set it.

55                      The name of the element will correspond to a method on the
            current focal point.
                        It will move the focal point to the result of that method.
```

```
        Example:
                In the below example,  we will set the focal point to the focal
        point associated with the
                name "Dialog Box".


                <ACTIONS>
                    <STR_ASSOCIATION FocalPoint="Dialog Box"/>

        -->
        <!ELEMENT STR_ASSOCIATION EMPTY >
        <!ATTLIST STR_ASSOCIATION FocalPoint NMTOKEN #REQUIRED >



        <!--
        ===============================================================================
        ========================
          Element: CALL_ACTION

          Summary: A Call Action represents a nested state machine.  When
        executed it will create a
                contexts that represents the state of that new state machine,
        and will begin executing
                on that new state machine.  The nested state machine will be a
        component state machine
                of the caller.  A state machine may composed of one or more
        nested state machines
                    using the call action construct.  This lets us decompose our
        state machine into reusable
                simpler state machines.


                When the nested state hits an end state it will return to the
        calling state machine
                where it will continue executing where it left off.


                All return values from nested state machines will be able to
        pass information back
                and forth, using a parameter passing technique that is
        available to all actions.

          Validation: The specified feature attribute (i.e. Nested State
        Machine) shall be a known feature.


          Example:
                In the below example,  we will call the state machine
        associated with the ParityChecker
                name.


                <ACTIONS>
                    <CALL_ACTION Feature="ParityChecker"/>
        -->
        <!ELEMENT CALL_ACTION EMPTY >
        <!ATTLIST CALL_ACTION Feature NMTOKEN #REQUIRED >



        <!--
```

126

```
     ==================================================================
     ========================
       Element: TRANSFORMATION

5      Summary: A Transformation Action will replace the current context's
     state machine logic with
               another logic specified by the Feature attribute.  A context
     represents the current
               state of a state machine.
10
       Validation: The feature attribute shall be validated against known
     features.

       Example:
15             In the below example,  we will be transforming our current
     context into a
               Origination Terminal state machine.


           <ACTIONS>
20             <TRANSFORMATION Feature="OrigTerm"/>
     -->
     <!ELEMENT TRANSFORMATION EMPTY >
     <!ATTLIST TRANSFORMATION Feature NMTOKEN #REQUIRED >


25


     <!--
     ==================================================================
     ========================
30     Element: LITERAL
       Summary: This is a constant value that is pushed on to the parameter
     stack of the engine.

       Example: A literal could be used in any action.  The startTimer action
35   takes two parameters,
               the number of seconds to set the timer for ( an integer ) and
     an event to post when
               the timer goes off.

40                     <ACTION Name="startTimer">
                         <LITERAL Name="IntLiteral" Value="3"/>
                         <LITERAL Name="EventLiteral" Value="Timeout"/>


     -->
45   <!ELEMENT LITERAL EMPTY >
     <!ATTLIST LITERAL Name NMTOKEN #REQUIRED >
     <!ATTLIST LITERAL Value NMTOKEN #REQUIRED >



50
```